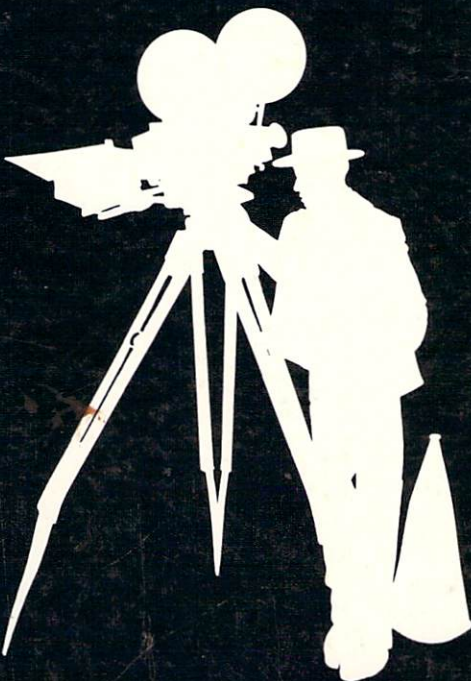




THE DIRECTOR™

PROFESSIONAL DISPLAY AND ANIMATION LANGUAGE FOR THE AMIGA™

A black silhouette of a director wearing a hat, standing next to a large movie camera mounted on a tripod. The director is holding a clapperboard. The camera has two large reels on top.

THE DIRECTORTM

SCREENPLAY
BY KEITH DOYLE

Copyright

Copyright 1987 Right Answers. All rights reserved. No part of this publication or the software supplied on magnetic media may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, in whole or in part, without the prior written permission of Right Answers, Box 3699, Torrance, CA. 90510, except for backup purposes by the end user.

Disclaimer

RIGHT ANSWERS MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

In no event will Right Answers be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the material described herein, even if advised of the possibility of such damages.

Information in this document is subject to change without notice and does not represent a commitment on the part of Right Answers.

Limited Warranty

Right Answers warrants the media on which the program is furnished to be free from defects in materials and workmanship under normal use for ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

Workbench Copyright 1985,1986,1987 Commodore-Amiga Inc. All Rights Reserved.

Trademarks

Amiga/Commodore-Amiga Inc.
DPaint/Electronics Arts
VideoScape-3D/Aegis
Grabbit/Discovery Software
DigiPaint/NewTek
The Director/Right Answers

*Our thanks to Richard Sexton, Howard Heard, Jim McInnis,
Deborah Jessen, and Vicki Eden for their help and support.*

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

CONTENTS

1. Welcome to the Director	
Philosophy.....	1-2
2. Getting Started	
Setting Up Your Disk.....	2-1
Subdirectories.....	2-2
Running the Examples.....	2-3
Configuring Your Working Disk.....	2-4
Typical Session.....	2-4
The Script File.....	2-5
Running The Director.....	2-5
General Sequence Operation.....	2-6
Distribution Of Finished Sequences.....	2-6
Building a Distributable Disk.....	2-6
Using This Manual.....	2-7
3. Introductory Tutorial Session	
The LOAD Command.....	3-1
The DISPLAY Command.....	3-2
The PAUSE Command.....	3-2
The GOTO Command.....	3-3
Page-Flipping.....	3-4
Enough Memory?.....	3-5
Keeping Files On Disk.....	3-6
Don't Fill the RAM: Disk With Commands.....	3-6
Using Fewer Memory Planes.....	3-6
Using Expansion Memory.....	3-7
More On Memory.....	3-11
Mixing Resolutions.....	3-11
Mixing Palettes.....	3-12
4. The Director's Language	
A Slideshow.....	4-1
Defaults.....	4-2
The REM Command.....	4-2
Using Colon to Shorten Scripts.....	4-3
Numbers.....	4-3
Variables.....	4-3
The Array.....	4-4
Simple Modifications of Program Flow.....	4-5
The GOTO.....	4-5
The GOSUB and RETURN.....	4-6

The IF ELSE and ENDIF.....	4-7
FOR/NEXT Looping.....	4-8
Text strings.....	4-10
Expressions.....	4-11
Double Buffering.....	4-13
Effects.....	4-16
Fades.....	4-16
Wipes.....	4-17
Characteristics of Wipes.....	4-17
Dissolve.....	4-20
Stencil.....	4-21
BLIT.....	4-24
Adjusting Speed.....	4-25
Text and Fonts.....	4-26
Text.....	4-26
Foreground and Background.....	4-27
Fonts.....	4-28
Configuring for Fonts.....	4-28
Determining Which Fonts Are Available.....	4-29
Using Fonts from Scripts.....	4-30
More Double Buffering.....	4-31
Drawing Commands.....	4-36
Color.....	4-36
Shaking the Bugs Out.....	4-37
Freeing Memory.....	4-38
Memory Monitoring.....	4-39
Ending Sequences.....	4-39
 5. Putting the Pieces Together	
Programming With the Director.....	5-1
 6. Advanced Techniques	
Library Routines.....	6-1
Video Tape Recording Your Sequences.....	6-1
String Handling.....	6-1
File Input and Output.....	6-3
OPEN.....	6-4
READ.....	6-4
WRITE.....	6-5
LOCATION.....	6-5
SEEK.....	6-6
CLOSE.....	6-6

Using File Input and Output.....	6-6
Input from the Keyboard.....	6-7
Keyboard Input Via CLI.....	6-8
Other Input.....	6-9
Single Keystroke Input.....	6-9
Mouse Input.....	6-9
Executing AmigaDOS Programs.....	6-11
Drawing Commands.....	6-11
Using BLIT to do Page Flipping.....	6-12
The BLIT Utility BLITUTIL.....	6-13
Changing BLITDEST and BLITMODE.....	6-14
Overscan Screens.....	6-14
ANIM File Playback.....	6-15
Bit Plane Manipulation.....	6-18

7. Sound and the Director

The Sound Module.....	7-1
Things To Think About.....	7-1
What is "Sampled Sound"?.....	7-2
Using the Sound Module.....	7-2
The LOAD sound command.....	7-3
The PLAY sound command.....	7-3
The QUIET sound command.....	7-4
The SLOWFADE sound command.....	7-4
The FREE sound command.....	7-5
The KILL sound command.....	7-5
Running Other Sound Programs.....	7-6

8. Command Reference

ABORT.....	8-1
ANIM.....	8-2
ARRAY.....	8-2
BLIT.....	8-3
BLITMODE.....	8-4
BLITDEST.....	8-6
BUFFERS.....	8-7
CENTER.....	8-7
CIRCLE.....	8-8
CLEAR.....	8-8
CLOSE.....	8-8
COLOR.....	8-8
COMPARE.....	8-9
COPY.....	8-10

CYCLE.....	8-10
DISPLAY.....	8-11
DISSOLVE.....	8-11
DRAW.....	8-12
DRAWMODE.....	8-12
ELLIPSE.....	8-13
ELSE.....	8-13
END.....	8-13
ENDIF.....	8-14
EXECUTE.....	8-14
FADE.....	8-15
FILL.....	8-15
FIXPALETTE.....	8-16
FOR.....	8-16
FREE.....	8-17
FREEFONT.....	8-17
GETKEY.....	8-18
GETMAP.....	8-18
GETMOUSE.....	8-19
GETPEN.....	8-19
GOSUB.....	8-19
GOTO.....	8-20
IF.....	8-20
IFKEY.....	8-21
IFMOUSE.....	8-21
INCLI.....	8-21
INPUT.....	8-22
LOAD.....	8-22
LOADANIM.....	8-23
LOADFAST.....	8-23
LOADFONT.....	8-23
LOCATION.....	8-24
MARGINS.....	8-24
MEMORY.....	8-25
MODULE.....	8-25
MOVE.....	8-25
NEXT.....	8-26
NEW.....	8-26
NEWFAST.....	8-26
OPEN.....	8-27
PALETTE.....	8-27
PAUSE.....	8-28
PEN.....	8-28

PLANES.....	8-29
POINT.....	8-29
POINTER.....	8-30
POSITION.....	8-30
PRINT.....	8-31
READ.....	8-32
RECT.....	8-32
REM.....	8-32
RESOLUTION.....	8-33
RETURN.....	8-33
ROTATE.....	8-33
SEEK.....	8-34
SETBLACK.....	8-34
SETFONT.....	8-35
SETLACE.....	8-35
SETMAP.....	8-36
SKIPANIM.....	8-36
SOUND.....	8-37
SPEED.....	8-37
STENCIL.....	8-38
STRING.....	8-38
STYLE.....	8-39
TEXT.....	8-39
TRANSPARENT.....	8-40
TROFF.....	8-41
TRON.....	8-41
WIPE.....	8-42
WRITE.....	8-43

Appendix A The ED Editor

Appendix B Using the CLI

CD.....	B-1
COPY.....	B-1
DATE.....	B-1
DELETE.....	B-1
DIR.....	B-1
DISKCOPY.....	B-2
ED.....	B-2
EXECUTE.....	B-2
FORMAT.....	B-2
INFO.....	B-2
LIST.....	B-2

MAKEDIR.....	B-3
RELABEL.....	B-3
SAY.....	B-3
TYPE.....	B-3

Appendix C Error Messages

Error Message Detail.....	C-1
Aborted.....	C-1
Argument Must Be String Array.....	C-1
Array Index Out of Range.....	C-2
Buffer Size Mismatch.....	C-2
Can't Open Diskfont Library.....	C-2
Can't Open _____.font.....	C-2
Divide By 0.....	C-2
Dest Must be Array.....	C-3
File Not Open.....	C-3
Invalid Cell Size.....	C-3
Math Stack Overflow.....	C-3
Math Stack Underflow.....	C-3
Missing Parameter.....	C-3
Next Without For.....	C-4
Nonexistent buffer.....	C-4
Nonexistent Line.....	C-4
Not An ANIM Buffer.....	C-4
Not a Version 1.2 Film File.....	C-4
No Screen.....	C-5
Out Of Memory.....	C-5
Parameter Not a String.....	C-5
Parameter Not a Variable.....	C-5
String Index Out of Range.....	C-5
Stack Overflow.....	C-6
Return Without Gosub.....	C-6
IF: Unmatched.....	C-6
Write Error.....	C-6

HINTS FROM OUR USERS

- It is important to remember that the first LOAD command automatically displays a picture. That picture can be permanently altered by DRAW and TEXT commands, or by DISSOLVEing or WIPEing another picture onto the screen. To avoid having your original picture changed, remember to COPY it to a NEW buffer before any operations affect it. (See NEW, COPY, and DISPLAY.)
- Remember that DISSOLVE, WIPE, and BLIT commands change what you *see* on the screen, without changing which *buffer* is being displayed. The DISPLAY command causes a different buffer to be the *displayed* buffer. (See buffer, DISPLAY, and explanation on pages 3-1 and 3-2.)
- In no way is it essential to master the entire manual. A handful of commands can produce exciting computer films. The “Probe Sequence”, for example, used very few commands: LOAD, DISPLAY, PAUSE, BLIT, WIPE, MOVE, DRAW, TEXT, PEN and FREE.
- Throughout the manual, examples of picture file names are in the form “:pictures/picfile”. On Amiga systems with hard disks, specifying the disk name in addition (i.e., “Disk:pictures/picfile”) makes it easier to install Director animations on the hard disk.

Send us your hints and comments.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

1

WELCOME TO THE DIRECTOR

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

1. Welcome to the Director

The "Director" is a slideshow/animation/scripting program that allows you to setup a series of display events to be sequenced by the Amiga including custom animations, graphics drawing functions, and the display of standard IFF image and ANIM files, as well as many other animations and effects.

The Director can be utilized for a variety of presentation tasks, including slideshows, video presentations, interactive games, educational presentations, and experimentation with the Amiga's powerful display capabilities.

Within the Amiga software community, most Amiga products make use of standard file formats where applicable. Most of these are the IFF standard formats (IFF stands for Interchange File Format). The Director is compatible with the IFF ILBM (InterLeaved BitMap) image files, and the IFF ANIM format (animation format). IFF ILBM files are picture or image files as generated by DPaint, DigiPaint, Grabbit and picture files that can be generated by DPaint, DigiPaint, Aegis Images, GraphiCraft and other programs. IFF ANIM files are animation files as generated by VideoScape-3D and other programs.

The Director can generate displays alone or in combination with IFF standard images and animations, and Amiga compatible fonts.

The Director will allow you to explore the Amiga's special graphics hardware with built-in commands that make it fun and easy.

The Director also provides a series of display effects such as dissolves, fades, and wipes, and provides several commands that will allow you to construct customized wipes and dissolves.

In addition, the Director can do a variety of animation effects, including page-flipping, and moving objects around the screen.

Philosophy

Unlike many other display and animation programs for your Amiga, the Director does not have a pictorial interface for configuring animations. We determined that the best way to retain the complete flexibility required to create sequence effects such as randomness and conditional looping, user interaction and other features, was to provide a "script"-oriented system.

In order to make a script oriented program as easy as possible to learn and use, we decided to have our script language conform where possible to the BASIC computer language. We chose BASIC because it is both easy to learn, many people already are familiar with BASIC, and a version of BASIC comes with your Amiga.

However, if you don't know BASIC, don't fear. This manual is designed to be easy to read and understand even for persons who may have never worked with a computer before. You do not need to be a programmer to be able to make good use of the Director.

The Director's script language provides a simplified set of BASIC-like commands, but at the same time differs from BASIC in that it has built-in features for reading IFF files, using text fonts, managing screens, using the Amiga's specialized graphics hardware and performing a variety of display effects.

2

GETTING STARTED



2. Getting Started

Setting Up Your Disk

As soon as you open your Director package, take your Director disk, and do the following:

- Check to see if the write protect tab on the disk is in the protected position. If you can see through the write protect hole, then the disk is protected. If the disk is not protected, WRITE PROTECT YOUR DISK.

- Next, BACKUP YOUR DISK. Use your WorkBench or CLI disk to do a duplicate or disk copy.

- Make a second copy of your Director disk for a working disk. You can keep an intact copy of the original disk so you can show the demo animations on the disk.

- Check out the demo animations by following the instructions in the next section on running the demos.

- Modify your working disk to contain any required startup-sequence parameters for your system. If you have a standard 512k system, or a standard system with only autoconfig RAM, then you won't have to do anything. If you have a hard disk, or want to put the Director files on a disk of your own configuration that you have prepared separately, copy the files "director", "projector" and "sound.mod" to your other disk, and follow the instruction on Configuring a Working Disk in the following section by that name.

- If you delete all the demo files on your working disk, you can copy some of your own picture files to it, to experiment with while trying out the various Director features. Initially, you might try selecting files of the same resolution, and perhaps even with the same or similar palette, until you are more familiar with the program and until you better understand the effects of using differing resolutions and palettes. Alternately, there are three files that are used in the example animation in part 5, that can be used to experiment with while trying various Director features. These are ":pictures/frame", ":pictures/images1", and ":pictures/images2".

Subdirectories

In the Amiga system, data on disks can be organized into what is referred to as "subdirectories". From the Workbench, these are referred to as "file drawers". From the CLI, they are called "subdirectories" and represent named "bins" that you can place files into. In our examples, picture files are always placed into a subdirectory called "pictures", in order to keep the disk from otherwise being a bit cluttered with files. A file within a subdirectory is specified as ":pictures/filename". For more information on subdirectories, consult the manuals that accompany your Amiga, or a book on AmigaDOS.

Running the Examples

The example programs can be run in one of two ways. You can either boot a workbench disk and just double click on the icons, or if you either boot the Director disk or a CLI disk, you can run the examples by using the Projector program. A file called TYPEME is automatically displayed when you boot directly off of the Director disk, or you can type "type TYPEME" from the CLI to see the message if you have booted from another disk. This message will describe the different example programs and how to run them. Note that if you are running a program that uses the sound module and have booted from another disk, you may need to do the command:

Assign mod: The_Director:options

so that the Director will be able to locate the sound module.

Configuring Your Working Disk

If you are setting up a fresh Director disk, you will need to know a couple of things about configuration.

If you are using sound, you must do an Assign mod: to the name of the directory that contains the sound.mod file. Otherwise, the Director will be unable to find the file, and an error will result. If your working disk name is named "workdisk", and the sound.mod file is copied to an "options" directory on your workdisk, the correct assign command will look like this:

Assign mod: workdisk:options

If you want the Director to create an icon to accompany your script, the file "directoricon" should exist in the directory used by the sound module and be assigned with the CLI command:

Assign dicon: workdisk:options

If you later decide you'd like to customize the icon being used by the Director, you can copy whatever icon you wish to the file "directoricon", and it will automatically be used as a template for further icons generated by the Director for your scripts.

Typical Session

The following steps illustrate the typical process involved in using the Director:

1. Create or modify a script file
2. Run the Director with the script file
3. Go back to step 1 until you're satisfied with the results
4. Package your finished sequence with the projector program for future viewing.

The Script File

The Director is driven with a "script" file, a standard Amiga text file which is a list of Director commands to be executed when the Director is run. In order to edit your script files, you must select a text editor program to use. The text editor Ed, is included with your Amiga in the c: directory. Instructions on how to use Ed are contained in the appendix. In addition, the Amiga Extras disk contains an editor called Microemacs. A manual for Microemacs is included on the Extras disk as a text file.

Most text editors are run from the CLI. The Director itself is run from the CLI, though finished animations can be run from the WorkBench.

In the appendix there is a quick reference section on CLI commands. If you would like to know more about the CLI, there are several books available on AmigaDOS at your local book or computer store that should be helpful.

Running The Director

To run the Director from the CLI, simply type:

```
director filename
```

where filename is the name of the script file to be used.

Using the script file, the Director will generate a "filmstrip" file. The file will be named the same as the script file name, but with .film added to the end of the name. After creating the .film file, the Director will proceed to display the sequence as instructed by the script file. Later, the resultant .film file can be "played" with the Projector program.

The Projector is run from the CLI with the command:

```
projector filename.film
```

or by double clicking on a .film file icon from the Workbench.

For convenience, The Director program can display .film files directly, just like the Projector program. The film file **MUST** end in .film to be run by either the Director or the Projector.

General Sequence Operation

Once a .film animation is running, you can use the right mouse button to pause and then restart the display. The left button can be used to abort the entire animation. The **ABORT** command can be used to modify this behavior (see command reference, part 8, for specific details).

Distribution Of Finished Sequences

The Projector does not need the original script file to run the finished display sequence. If you only distribute the Projector and the .film files (and any associated IFF files involved) you can minimize unauthorized modifications of your sequences. On the other hand, you can distribute your script files along with your .film files and the Projector, to allow other people who may have their own copies of the Director to modify your sequence.

The Projector program and the sound.mod module are copyrighted, but can be freely distributed provided they are not included in a product that is for sale. ".film" files can be distributed freely in either case. The Director program is copyrighted, and duplication is permitted only for personal archival purposes.

If you build a sequence and you want to include it and the Projector in a product that is for sale, we request that you include somewhere in the product's manual the statement "This product includes display sequences that were generated using the Director by the Right Answers Group." In addition, we also request that you send us a copy of the product software and manual for our files. If your product includes hardware, please contact us for an equitable arrangement.

Building a Distributable Disk

For any animations you intend to distribute, please be sure that only the "projector" program and the "sound.mod" module are contained on the disk, and not the "director" program.

There are several ways you can accomplish this. There may be a strong tendency just to copy the disk and delete the "director" file. There are two problems with this method. One, it is possible to recover files that have been deleted with certain disk utilities. Two, at the time of this writing, it is not legal to distribute a disk with the Commodore system files on it, such as the commands in the "c" directory, any library files, etc. Commodore may change this policy in the future, but until they do, you should only distribute your animations to the public on a disk all by themselves (and with the projector and sound.mod files where required).

You can create a disk with your animations on it as follows:

1. Format a new disk using the Amiga format utility
2. Copy the projector and sound.mod files to the disk
3. Copy all required image, anim, and sound files to the disk
4. Copy your script file and it's associated icon file to the disk

If a script is using sound, you need to make sure the Director is able to find the sound.mod module, by making sure that an Assign mod: command has been done. If you find some difficulty in making sure the Assign mod: command has been done, you can include a Director EXECUTE command in your script of the form:

```
EXECUTE a,"Assign mod: diskname:options"
```

where "diskname" is the name of your disk to be distributed. This will cause the correct assign to happen automatically when your script is run. This command must precede the MODULE command that runs the sound module.

Using This Manual

The rest of this manual is organized into Introductory, Intermediate, Putting The Pieces Together, Advanced and Reference chapters.

If you have very little experience with a computer, start with the Introductory chapter. If you have played with AmigaBasic or some other Basic before, you can scan over the Introductory chapter, and concentrate on the Intermediate chapter which will cover the areas where the Director differs from standard Basics.

"Putting the Pieces Together" takes you through the creation of a short animation as an illustration of many of the most often-used commands.

Don't feel you have to understand ALL of the Director's commands before you will be able to do anything. In fact, most people will find that there are many commands they will never need because the types of sequences they are interested in never require certain features. As you learn the basic commands, you can begin to try combining them in different ways, to accomplish specific tasks in your animations.

Ultimately, it is up to you to have the idea for a "story" you may want to tell, or a presentation you may want to give using the Director as a tool. As you become more familiar with the Director's basics, it probably will be beneficial to read the manual through, with no intention of trying ALL the commands right away. As you begin to work on specific sequences, and specific situations present themselves, cursory knowledge of all the capabilities of the Director will help you to decide if the best solution to a particular implementation problem is entirely contained in the commands you have used, or may benefit from further investigation into commands you haven't tried yet.

After you feel comfortable with the Director's basic functions, and/or as you need to learn about more complicated Director possibilities, you can explore the advanced chapter for various hints and techniques for doing more advanced special effects.

If at any time, you want to know more about a specific command, check the Command Reference where you will find them all in alphabetical order, or the Index where you will find other discussion of the command in the manual. A few commands may not be provided with explicit examples in the beginning chapters, but you will find specific details in the Command Reference, and other examples in the example scripts provided on the Director disk.



3

INTRO TUTORIAL SESSION



3. Introductory Tutorial Session

A script file consists of a sequence of commands that instructs the Director on what to do and in what order. Let's start off with the commands you will probably use in just about all of your sequences, the LOAD, DISPLAY, and PAUSE commands.

The LOAD Command

A LOAD command will look something like this:

```
LOAD 3,":pictures/tree.pic"
```

The LOAD command has two "parameters". Parameters are always separated by commas. The second parameter, ":pictures/tree.pic", is the name of a standard Amiga IFF picture file to be loaded into the computer's memory from a disk. The first parameter, 3, is the number of a screen buffer into where the picture is to be loaded. A buffer is simply a storage area in the computer's memory.

The picture file name here is specified starting with a colon (:), which means "from the currently selected drive". You could also specify the complete drive name, as in "df1:pictures/tree.pic", or include the actual disk's name, as in "mydisk:pictures/tree.pic". This last method allows the Director to find the image file no matter which disk is in which drive.

When you first start up the Director there are 30 available screen buffers, numbered 1-30 (more can be configured using the BUFFERS command described later). Only when you LOAD a picture into a buffer is a buffer actually using internal memory space for your picture.

By using buffers to store your pictures, you can preload several pictures at the beginning of a sequence. This will allow them to be displayed and manipulated later without the pauses caused by disk reads. Here's how you can load three pictures into computer memory using the LOAD command:

```
LOAD 1,":pictures/landscape"  
LOAD 2,":pictures/tree"  
LOAD 3,":pictures/house"
```

The first LOAD command causes a new screen to be setup with the picture specified displayed. Later LOAD commands will load into their respective screen buffers, but will not change the observed display. Other commands such as DISPLAY can then be used to incorporate these new images into a presentation.

The DISPLAY Command

So how do we display a screen buffer other than the one we loaded first? We use the display command:

DISPLAY 2

This causes the picture stored in screen buffer 2 to be displayed. The previous displayed buffer will remain in its buffer, but will no longer be displayed.

The PAUSE Command

Many times while running Director sequences, you will want to pause to give the viewer time to see what is on the screen, or to adjust the timing of your sequences. Use the PAUSE command:

PAUSE 20

The PAUSE command will cause the Director to wait a period of time before executing the next instruction. The single parameter for the PAUSE command, in this case 20, is a count of tenths of a second. 20, then, will result in a pause for two seconds. You can pause in smaller increments by using the SPEED command which will be described later.

So let's build a sample Director script from the commands we know so far:

```
LOAD 1,":pictures/landscape"  
PAUSE 50  
LOAD 2,":pictures/tree"  
DISPLAY 2  
PAUSE 50  
LOAD 3,":pictures/house"  
DISPLAY 3  
PAUSE 50
```

This will give us a simple slideshow that will automatically exit when finished.

The GOTO Command

Let's say we wanted to see those pictures again. Perhaps we want to produce a presentation that loops over and over displaying our same three pictures.

We can do that with the GOTO command, like this:

```
10      LOAD 1,":pictures/landscape"  
        PAUSE 50  
        LOAD 2,":pictures/tree"  
        DISPLAY 2  
        PAUSE 50  
        LOAD 3,":pictures/house"  
        DISPLAY 3  
        PAUSE 50  
        GOTO 10
```

The 10 up in front of the first LOAD command is called a line number. You can assign any numbers you wish to any line, as long as you don't re-use the same number twice in the same program. We've numbered that first line, so we can "reference" it later in the GOTO command. The GOTO command tells the Director to "jump" elsewhere in the instruction script, to a place that is out of the normal sequence of commands. So in this case, the GOTO command tells the Director that when it gets to the end of the script, to just go back to the top and do it all over again.

How do we then tell the Director to quit when we have told it to loop forever? The left mouse button can be used to signal the Director to quit. The right mouse button will tell the Director to pause and then if hit again, will tell the Director to resume. This way if you would like to stop on a particular image, you can pause it as long as you want by using the right mouse button. The function of the left mouse button can be modified with the ABORT command which is described later.

The second time through the loop, in our example, we are loading a picture into buffer 1 that has ALREADY been loaded into that buffer. The same with pictures 2 and 3. We really don't have to keep loading them again and again.

We can adjust our script to take that into account:

```
10      LOAD 1,":pictures/pic1.pic"  
        LOAD 2,":pictures/pic2.pic"  
        LOAD 3,":pictures/pic3.pic"  
        DISPLAY 2  
        PAUSE 50  
        DISPLAY 3  
        PAUSE 50  
        DISPLAY 1  
        PAUSE 50  
        GOTO 10
```

Here we have loaded all of our pictures first, and then we do all of the displays so we don't have to include the LOADs in our loop. The Director will display our three pictures until we signal "quit" with the left mouse button, without having to access the disk drive after the first pass.

Since we no longer have to wait for the disk drive, we can speed up our display considerably. If we adjust all of the PAUSE parameters, say, down to 1, the pictures will change 10 times a second. If we remove the PAUSEs entirely, the pictures will go by so fast, we may have a hard time even seeing what they are.

Page-Flipping

A fast "flip" of displayed pictures as in the previous example is called "page-flipping". It is similar to the idea behind cartoon "flip" books you may have seen, where each page has a slightly different cartoon frame on it, and when you ruffle the pages with your thumb, you

can see an animated cartoon.

If our pictures 1, 2, and 3 are related in some way, our display with the PAUSEs adjusted to be very short can cause the same type of animation effect as a flip book, or even as a motion picture film, which operates on the same basic principle. With a motion picture film individual frames go by, or "flip", at 24 frames per second. The Director can flip frames at over twice that rate.

We will have to preload a lot of frames to do much in the way of animation at such high frame rates. Fortunately, it turns out that a lot of animation doesn't actually need to run at such high rates. Much cartoon animation you see on TV runs at somewhat less than the 24 frame film rate. That is, even though the images are flashing at us at 24 frames per second, they are only changing at some lower rate, perhaps eight frames per second or so. It is quite possible to generate effective page-flipping animation at similar rates.

Enough Memory?

In addition, the Amiga has a limited amount of display memory, called "chip RAM" that display buffers can reside in. How many buffers can we have? It depends on the resolution of the picture we are trying to display. In a 512k Amiga, some of the chip RAM is taken up by the operating system, some is taken up by the Director program, and only what is left is available for screen buffers. A low resolution 320x200 image using 32 colors (5 memory planes) requires 40,000 bytes of memory. This would probably only allow us somewhere in the range of 8-10 frames in memory. Not a lot for much of an animation!

It may turn out that our animation doesn't need all that much memory. Page-flipping tends to use a lot of memory, but there are a lot of other things that don't. For an example, look at the "tvgal" high resolution animation on the Director disk. That animation can run in 512k and still give a presentation that lasts a minute or so. It all depends on what we are trying to do. Presentations that generate a lot of text, could go on for hours without repeating and without accessing the disk.

So how can we provide for more space for our animations when we do need more memory? There are several techniques:

Keeping Files On Disk

One way to minimize your usage of memory is to not preload all of your images into memory ahead of time. You can load them one at a time as you need them. One drawback of this method, is that it takes several seconds to load an image, causing a pause which is not always appropriate for a sequence you may be preparing.

Don't Fill the RAM: Disk With Commands

Every time you copy files into the RAM: disk, this borrows internal computer memory that could otherwise be used by screen buffers. Unless you have a lot of extra memory, one way to increase the memory available for screen buffers, is to not copy your AmigaDOS command files into RAM:

If you don't understand what this means, then you shouldn't have to worry. When you get the standard Amiga software from Commodore, it is configured to not copy command files into RAM:. Many people do this in order to speed up access to commonly used AmigaDOS commands. The Director master disk is also configured to use only as much RAM: as absolutely required.

In addition, if you are running either the WorkBench or various utilities such as the CLOCK, POPCLI, PM, etc., these all take up memory.

Using Fewer Memory Planes

One way we can get more frames in memory for page-flipping is to decide to use fewer colors, and thereby cut down the number of memory planes required for an image. DPaint II will ask you at the startup screen how many colors you wish to use, and will adjust the color palette accordingly. By designing your pictures in DPaint with fewer colors, you can cause your resultant pictures to occupy less memory and thereby be able to LOAD more frames into chip RAM.

A single plane at 320x200 requires 8000 bytes of memory. So you could get close to 50 frames in memory.

On the other hand, a 640x400 16 color display requires 128,000 bytes. You will only be able to get at most 3 frames in a 512k system at that resolution.

Using Expansion Memory

If you don't have expansion memory, you can skip this part as it won't apply to you.

If you have extra memory, there are things you can do to make use of this memory. Right away you have a little more room, as the Director program will load into your expansion memory automatically if the memory is configured, thereby using less of your valuable CHIP memory.

You can use the LOADFAST, NEW, NEWFAST and COPY commands to load pictures into expansion RAM (sometimes called FAST memory) and move them between FAST and CHIP memory during your display.

The LOADFAST command takes exactly the same kind of parameters as the LOAD command, except that it will load the pictures into FAST memory. Don't try to DISPLAY them from there though, as the Amiga hardware is not able to display out of FAST memory.

The NEW and NEWFAST commands are provided to create a new buffer without loading it from disk. With these commands, you can create extra screen buffers for copying images.

The NEW and NEWFAST commands have two parameters, and look like this:

```
NEW 5,3  
NEWFAST 5,3
```

The first parameter is the number of the new buffer you are creating. The second parameter is the number of an existing buffer to use as a "model". The new buffer will be created with a screen resolution and color palette identical to that of the "model" buffer

The COPY command can be used to copy full screen information between buffers of the same resolution.

The COPY command looks like this:

```
COPY 3,5
```

The first parameter is the buffer number to copy from, and the second parameter is the buffer to which to copy.

Now that we have these new commands, let's construct a script that uses them:

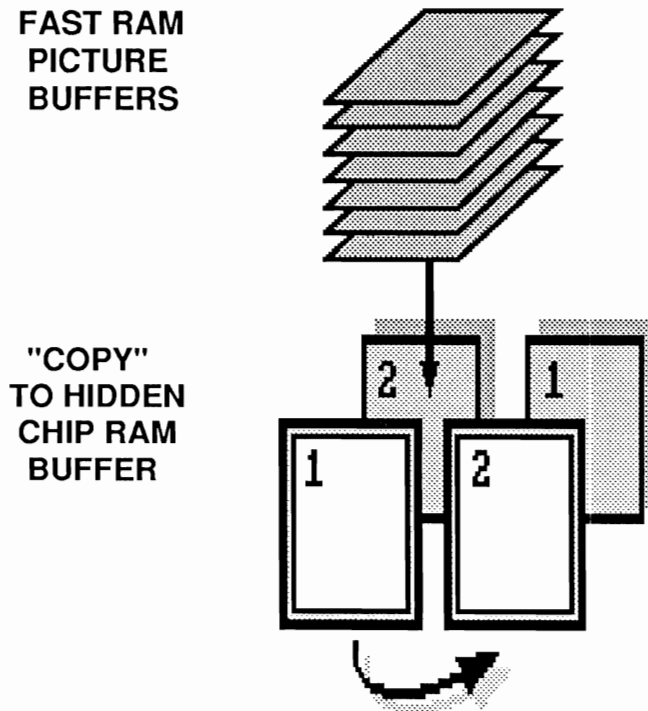
```
10      LOADFAST 3,":pictures/pic1.pic"
        LOADFAST 4,":pictures/pic2.pic"
        LOADFAST 5,":pictures/pic3.pic"
        NEW 1,3
        NEW 2,3
        COPY 3,1
        DISPLAY 1
        PAUSE 2
        COPY 4,2
        DISPLAY 2
        PAUSE 2
        COPY 5,1
        DISPLAY 1
        PAUSE 2
        COPY 3,2
        DISPLAY 2
        PAUSE 2
        COPY 4,1
        DISPLAY 1
        PAUSE 2
        COPY 5,2
        DISPLAY 2
        PAUSE 2
        GOTO 10
```

Note that we alternately use CHIP buffers 1 and 2 to hold our image. The COPY command is not 100% instantaneous. If you copy to a screen you are displaying, you will see a slight "wipe" effect as the data is copied in from top to bottom. In order to minimize that possibly undesirable effect, we can copy to a second non-displayed buffer, then display that buffer when we are done copying.

This is the basic concept behind "double-buffering", a term you will hear more about, as it is a very useful way to insure that the display modifications happen cleanly and simultaneously. What we do, is work on a non-displayed buffer, until it has been prepared for display, then display it and use the previously displayed buffer as a working buffer for our next display.

Let's look at the illustration on the following page to help make this clearer.

ANIMATING FROM FAST RAM



The white rectangles at the bottom represent the buffer being displayed on the screen, while the gray rectangles behind them represent the "hidden" chip buffer. We always copy to the buffer that is hidden, and after the copy has finished, we display it. This exchanges our hidden and displayed buffer, as the previously displayed buffer now becomes hidden.

When copying high resolution full color screens to or from FAST ram, in some Amiga configurations the Amiga main processor is slowed down by the display processors. If your FAST ram is truly no-wait-state FAST ram, then it will run at full speed. Even at full speed, copying of high resolution 640x400 images is not quite fast enough to do effective page-flipping unless perhaps you stick with single plane two color images. If, as in some systems, your FAST ram speed is tied to your CHIP ram, then you will find that COPYING to and from FAST buffers is slower yet.

Note that in our last example, we went through the pictures twice before we went to the top of the screen. Why? Because we alternately used CHIP buffers 1 and 2 to display our images, and we have an odd number of images which causes us to end up with the wrong buffer being displayed at the end of the loop then what we want at the top. We can fix that so we don't have to duplicate all those COPYs and DISPLAYs. In order to do that, we have to learn a little more about variables and expressions in the next chapter.

More On Memory

As you continue to read through the manual, we'll tell you about other techniques for keeping the memory used by your presentations down to a manageable level.

Mixing Resolutions

The Director will allow you to load IFF files of varying resolutions. Not all effects will be useful across screens of differing resolutions. The WIPE and DISSOLVE command may not provide useful results when performed between screens that are of differing resolutions. With HAM screens, many effects may cause relatively bizarre looking results. HAM is a complicated display format, and BLIT, WIPE, and DISSOLVE will interact with it in varying ways. That is not to say that some of these effects may not be useful.

If the images have a different number of bit planes, only the fewer number of bitplanes will be transferred in a BLIT, WIPE, DISSOLVE, COPY or similar commands. In some cases this can be useful, if you want to transfer images that use a subset of the screen's total colors to the screen, and are working with limited memory where the fewer bit planes you use in an image, the less memory it will consume.

The number of bit planes in a picture, is directly related to the size of the color palette. Here is a table of the relationships:

number of colors	number of bit planes
HAM 4096	6
64	6
32	5
16	4
8	3
4	2
2	1

You can determine pretty closely how much memory a given resolution will consume, by multiplying the X and Y resolution together, dividing by 8, and multiplying by the number of bit planes. Adding about 200 bytes afterwards for good measure will roughly compensate for other non-resolution dependent overhead that is involved in maintaining the in memory buffer and associated color palette.

Be aware of the fact though, that if your memory has been fragmented by fonts or drivers that have been loaded by previously run programs, that you will probably not be able to fill your available memory up to the last byte. The Amiga system allocates memory in varying size chunks depending on what the program needs at any given time. If the program requests a large block, and though there is enough total available memory for such a block, but it is fragmented (spread around the system) so the system cannot provide that amount of memory as a single chunk. Each individual bit plane of an image file must be a single continuous chunk of memory.

Mixing Palettes

With many WIPE effects, parts of two different picture files may be on screen simultaneously for a short period of time. If the palettes for the two pictures are not the same, you will find that the colors of one or the other picture will be incorrect. The Amiga hardware can only support one palette at a time for screens that contain such images. In

some situations, differing palettes can be useful, in others it may be distracting. In our experience, we've found that is usually more effective to use the same palette while preparing images for a presentation where a lot of wipes or dissolves are desired. Varying only selected colors in the palette from screen to screen can provide for various special effects.

The general rule for palettes is, once an effect command such as WIPE or DISSOLVE finishes it's operation, it will then update the current screen's palette to match that of the buffer the effect is WIPEing or DISSOLVEing from. If this is undesirable, you can use the PALETTE command to switch the palette beforehand, use the FIXPALETTE 1 command to turn off the updating of palettes at the end of these commands, or make sure that the color palettes for the two images involved are compatible.

Some ANIM files may have a fixed palette which you might want to be compatible with. Use DPaint or other paint package, to load the first frame of the anim file just as if it was a normal image file, and use the resultant palette to create your compatible images. Remember not to re-save the image with the same name as the ANIM file, or you will lose the ANIM file and just have the first frame saved as a standard image file.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

4

THE DIRECTOR'S LANGUAGE

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

4. The Director's Language

The Director's language is a full fledged programming language, similar to BASIC, but with a series of powerful display commands oriented towards graphic presentations and animations. Everything from a simple slideshow, to animations, games, and interactive presentations can be created.

Now perhaps you're saying "What! Programming! I have to do programming?" Let's think about this for a minute. What is a program anyway? A television program is a planned sequence of events that takes place over time. Someone usually has to outline that sequence of events up front, in a "script" that is a list of what things to do and in what order. A computer program works basically in the same way, though it actually has several features that make it easier. In a computer program, you can re-use segments of your program several times without reentering the commands, you can specify loops that cause sections to be done over and over a specified number of times, as well as many other things.

So programming is not really anything to be afraid of. If you have worked with any of the public domain slideshow programs, the Director presents basically the same concept, but with significantly more capabilities.

A Slideshow

We've covered basic page-flipping effects in the previous chapter, let's talk about doing a simple slideshow presentation, and expand it more from there.

Here is an example slideshow script:

```

    REM Example script
    REM This script will load several pictures
    REM and then slowly sequence them on the
    REM display until aborted by an operator
    REM mouse click
    LOAD 1,":pictures/pic1"
    LOAD 2,":pictures/pic2"
    LOAD 3,":pictures/pic3"
100  PAUSE 20
    DISPLAY 2
    PAUSE 20
    DISPLAY 3
    PAUSE 20
    DISPLAY 1
    GOTO 100
```

Director commands can be entered either in lower or upper case. All examples will be in uppercase in this manual. Leading spaces before commands are not required in script files, but are used in the examples for clarity.

Defaults

In several commands, the number -1 is used to specify a default value, especially when specific parameters are used most of the time, or are otherwise particularly difficult to remember. Instructions on the individual commands in the Command Reference chapter will point this out in more detail.

The REM Command

Note that we have introduced a new command here, the REM command. REM is short for "remark", and is a way of including comments in your scripts so that you can remind yourself or others later, of what you were doing and why. REM commands are completely ignored by the Director so whatever you place on the line following the REM command, is just for your own information. We will use REM commands in our examples to help you to understand what is going on.

Using Colon to Shorten Scripts

In the examples we've seen so far, each command takes up one line in the file. In some cases, we might find it useful to put more than one command on the same line so that we can combine simple statements and be able to see more of the program on the screen at a time while we are editing it.

The Director allows you to combine commands on the same line by separating the commands with the colon symbol : .

For example:

```
DISPLAY 2:PAUSE 20:COPY 2,1
```

You can put as many commands on a line as you want as long as the line does not exceed 100 characters.

Numbers

All numbers represented in the Director are integers, that is, all numbers are positive or negative whole numbers, and are restricted to the range -2147483648 to 2147483647. This corresponds to a 32-bit or 4-byte signed representation. Commas are not allowed in numbers to separate every thousands position, as the comma is used to separate individual command parameters. There may be extra restrictions on numbers when using the "@" array, (see the section on the array for more details).

Variables

The display language supports numeric variables which can be up to eight characters of upper or lower case alphabets. Numbers and symbols are not allowed in variable names, and variables can not be named the same as a Director command.

The following are valid variable names:

```
AMOUNT
position
Fred
FrEd
```

Lower case and upper case are NOT considered the same within variable names. The two variables Fred and FrEd above, are considered two separate variables in the Director language.

The following are illegal variable names:

```
count6      (numbers not allowed)
Actualcost  (too many characters)
Date/Tim    (symbols not allowed)
end          (same name as a Director command)
```

Variables can be used to save numeric values for later processing within Director instructions. Instructions that may be re-used during a loop, can contain variables which can be modified to cause different things to happen each time through a loop.

A variable is given a value by using the equal sign:

```
COUNT=5
TOTAL=3+7*AMOUNT
```

The Array

In addition to variables, there is an array. The array is specified with the "at" sign, "@". The array is like a big collection of variables, all referenced with a number instead of a name. Each variable in an array is usually referred to as a "cell". Cells are referenced by number, and the reference number can be the result of a computation based on other array cells, or variables, or explicit numbers, in any combination. The reference number is usually referred to as an "index". An "index into the array" is a number that identifies a specific "cell" location in the array.

The array must be dimensioned with the ARRAY command which includes the size of the array (number of cells), and how big each cell is, in bytes. Adjusting the size of the cell allows you to tradeoff the amount

of memory used by the array for the number precision needed by the numbers stored in the array. The array can be as big as the memory available to the program in your Amiga when it is run. (See the Command Reference chapter for more details on the ARRAY command). Once dimensioned, the array elements can be assigned values as in these examples:

```
@(5) = 7
@(I+9) = T
```

And referenced within expressions like these:

```
N = 4*@(5)
PRINT "The answer is: ";@(I+5)
```

The "@" array is defined by the ARRAY command to be either a 1,2 or 4 byte array. This signifies the memory size of each array element. If you only need small numbers, 0-255, use a 1 byte array. If you need numbers in the range -32768 to 32767, use a 2 byte array, and if you need numbers up to the maximum integer range of -2147483648 to 2147483647, use a 4 byte array.

Simple Modifications of Program Flow

Director sequences are basically of the form:

```
instruction 1
instruction 2
instruction 3
instruction 4
instruction 5
```

In other words, a sequential list of Director "instructions" or functions to be performed. The GOTO and GOSUB/RETURN instructions are simple ways to modify the "flow" of the program to produce more efficient results, or to provide loops.

The GOTO

There are several Director "instructions" that can cause this sequential nature to be interrupted for a variety of useful reasons.

For example:

```
10      instruction 1
        instruction 2
        instruction 3
        instruction 4
        instruction 5
        GOTO 10
```

The "10" on the first line, is called a "label", or a "line label". This allows you to reference the "labeled" point in your scripts with other commands, such as the GOTO command illustrated here. This script will execute the five specified instructions and then, via the GOTO command, turn around and do them all again, endlessly (or until aborted by a mouse input).

The GOSUB and RETURN

Another way of modifying program flow, is with the GOSUB and RETURN commands:

```
        instruction 1
        GOSUB 10
        instruction 2
        GOSUB 20
        instruction 3
        GOSUB 10
        END

10      instruction 4
        instruction 5
        RETURN

20      instruction 6
        RETURN
```

The GOSUB command will:

1. Remember where we are in the script for later reference,
2. Go to the specified line (specified by label),
3. Execute instructions from that line on, until a RETURN is encountered,
4. "Return" back to where we remembered in step 1, and continue on from there.

This allows you to re-use segments of the program several times throughout your presentations, among other things.

The IF ELSE and ENDIF

Sometimes it is desirable for a script to make decisions based on the results of computations or of external input. The IF command allows you to test variables and to cause different things to happen based on the results of the tests.

Here is an IF statement example:

```
instruction 1
IF T<8
    instruction 2
    instruction 3
ENDIF
instruction 4
```

The use of IF in the Director is a little different from the IF found in most Basics. The IF statement contains an expression, the value of which is tested for zero. If the result of the expression is zero, then the Director will skip the commands between the IF and the ENDIF and continue with any commands that may be following the ENDIF. If the result of the expression is non-zero, then the Director will execute the commands following the IF statement and then continue on with whatever commands may be following the ENDIF.

Here, after instruction 1 is executed, the variable T will be tested to see if it is less than 8. If it is, then instruction 2 and 3 will be executed, and then instruction 4. If it is not, then only instruction 4 will be executed. Instructions 2 and 3 have been indented simply for clarity and is not a requirement.

Here is another IF example:

```
instruction 1
IF T>3
    instruction 2
ELSE
    instruction 3
ENDIF
instruction 4
```

Here, after instruction 1, if T is greater than 3, then instruction 2 is executed, and then instruction 4. If T is not greater than 3, then instruction 3 is executed, then instruction 4. In other words, "If T is greater than 3 do 2, ELSE do 3".

IF statements can test a variety of conditions for either true or false. Any expression can be used in an IF statement. If the expression evaluates to a zero it is considered false, and if non-zero, is considered true. Several conditions can be tested at a time:

```
IF (T<7) & (T>3)
```

This says, "If T is less than 7 AND is greater than 3, ..."

FOR/NEXT Looping

Sometimes it may be necessary within a sequence to do a series of instructions several times over before continuing to later instructions. You could simply repeat the instructions in the script several times, or, you can use the FOR/NEXT loop to make this easier.

The syntax of the FOR loop is something like this:

```
FOR V=1 TO 7
  instruction 1
  instruction 2
NEXT
instruction 3
```

This FOR statement says, "Let's do a loop. During the loop, the variable V will count from 1 to 7." The NEXT statement signals the end of the loop. So this means that instruction 1 and 2 will be executed 7 times before instruction 3 will be executed. During those 7 times, the variable V will contain a different value each time through the loop. In some sequences, this variable can be used to compute parameters for instructions that can cause the program to do slightly different things each time through the loop.

In the above example, any variable name can be substituted for V, and any legal numbers can be substituted for the 1 and 7.

Sometimes it is desirable for the FOR loop variable to count from its minimum and maximum by something other than 1's. Sometimes you might like to count by 2's, or by 100's etc. You can do this by adding the optional STEP parameter to a for loop.

Like this:

```
FOR V=1 TO 7 STEP 2
```

And sometimes you may want to count down instead of up. If the step parameter is negative, then you can count down:

```
FOR V=7 TO 1 STEP -1
```

Actually, if the STEP amount is to be either 1 or -1, you don't need to include it, as the Director will assume 1 or -1 if no STEP is given. 1 will be assumed if the second number is larger than the first, -1 otherwise. In addition, other expressions or variables can be specified in a FOR loop:

```
FOR V=T TO L*5 STEP N
```

FOR loops can be nested, that is, you can have a loop inside of another loop:

```
FOR V=1 TO 7
  instruction 1
  FOR T=1 TO 3
    instruction 2
    instruction 3
  NEXT
NEXT
```

The first NEXT matches the second FOR, this is the inner loop. The first FOR and the last NEXT specify the outer loop.

Text strings

Text strings can be stored and manipulated in the "@" array. Each text character in a text string will take up an entire "@" array element, no matter what size element is specified by the ARRAY command. For this reason, if a lot of string manipulation is desired, and little or no numeric use of the "@" array is required, setting the @ array to 1-byte is the most space efficient.

Strings are always stored with a 0 at the end. When a maximum string width is specified in a command, the maximum number of cells in the @ array is actually width+1, as the ending 0 is not counted. Keep this in mind when reserving space in the @ array for string storage. In addition, the Director cannot manipulate strings that are larger than 100 characters long. There is no specific limit on the number of strings you can have. The Director language provides several commands to make use of strings to input, compare, and output them. See the String Handling section in chapter 6, or the INPUT, COMPARE and STRING commands for specific details.

Strings can be specified in program statements either explicitly, with quotes around them like this:

```
"This is a string"
```

Or as strings contained in the @ array like this:

```
$(20)
```

```
$(I+5)
```

Where the expression in parenthesis will be used as the position within the @ array where the desired string starts. The dollar sign is used to indicate a string contained in the @ array.

You can convert a string of numeric digits into a numeric variable, by using the equals sign as follows:

$$N = \$ (0)$$

Expressions

Any numeric command parameter can be an expression. An expression can be a single numeric constant, or a variety of mathematical statements.

Examples of expressions are:

500
-2
3*7
A/5
(B+C) /9
?50+7&15

The valid arithmetic operators are:

- * Multiply
- / Divide
- % Modulo (also known as remainder, A%B= the remainder of A/B)
- + Addition
- Subtraction
- ! Absolute value (!-1 evaluates to 1)
- < Less than (evaluates to 1 if less than, 0 if not)
- > Greater than (evaluates to 1 if greater than, 0 if not)
- = Equals (evaluates to 1 if equals, 0 if not)
- # Not Equals (evaluates to 1 if not equal, 0 if equal)
- & Logical AND (A&B evaluates to 1 if both A AND B are non-zero, 0 if not)
- | Logical OR (A|B evaluates to 1 if either A OR B are non-zero, 0 if not)
- ^ Exclusive OR (A^B evaluates to 1 if either A or B but NOT BOTH are non-zero, 0 otherwise)
- ? Random number (?50 will evaluate to a random number between 0 and 49)
- ~ MAX operator (A~B evaluates to A if A>B, to B otherwise)
- _ MIN operator (AB evaluates to A if A<B, to B otherwise)

Variables can be included in expressions at any point where a constant value would otherwise be valid, i.e. an expression like 500*2+7 can also appear like A*2+7 or 500*A+7 or 500*2+A or A*B+7 etc.

Here are some specific examples:

10*VAL+6	(multiply VAL times ten and add six)
?200	(pick a random number 0 through 199)
?(SIZE+8)	(pick a random number 0 through SIZE+7)
VAL%10	(the remainder of VAL divided by 10)
!(VAL-50)	(the absolute value of VAL-50)
(VAL<20) (VAL>50)	(a 1 if VAL < 20 or VAL > 50)
VAL~20	(20 if VAL < 20, VAL otherwise)
5=VAL	(a 1 if 5 = VAL, a 0 otherwise)
100_VAL	(100 if VAL >100, VAL otherwise)

Double Buffering

Let's do a slideshow example, applying some of the things we have learned:

```

    REM a disk based slideshow
    SETBLACK 1
    BUFF=1
10   LOAD BUFF,":pictures/flower.pic"
    GOSUB 20
    LOAD BUFF,":pictures/tree.pic"
    GOSUB 20
    LOAD BUFF,":pictures/rock.pic"
    GOSUB 20
    LOAD BUFF,":pictures/fish.pic"
    GOSUB 20
    GOTO 10

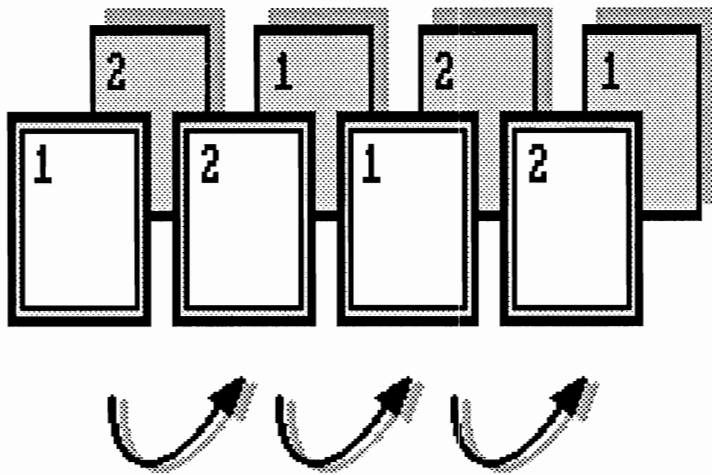
20   DISPLAY BUFF
    SETBLACK 0
    BUFF=3-BUFF:REM switch to other buffer
    PAUSE 20
    RETURN
```

In this example, we use a variable called BUFF that will toggle between a 1 and a 2 that is used by the LOAD command to specify the buffer number. By using the GOSUB 20 subroutine, we can re-use the code starting at line 20 after every LOAD. The subroutine starting at line 20, will then display the buffer just loaded, do our pause, and toggle the BUFF variable to equal the number of the buffer to be used by the next LOAD command. (for more on subroutines, see the previous section on program flow.

This is the basic mechanism for double-buffering that can be used with the Director. The idea behind double buffering is that you use two buffers, and are displaying one while you are preparing the other as the next display. Though this slideshow is a simple example of this, the same basic concept can be used when using the drawing commands, or the blitter commands to do animations, all using the same basic concept as demonstrated in the subroutine at line 20 above.

The illustration on the following page may help clarify what is going on while double-buffering.

DOUBLE BUFFERING METHOD



The boxes in the foreground represent the buffers that are being displayed on the screen. As we change images, the buffer we are displaying changes from 1 to 2 and back again. At the same time, the buffer we are not displaying is hidden (represented in grey in the diagram). While double-buffering, we always load our new image (or generate a new image with other commands) on to the buffer that is hidden (non-displayed).

This can be a confusing to keep track of. It is for that reason the GOSUB 20 in our last example was created to help keep track of things automatically. Using the GOSUB at line 20, it takes care of which screen should be displayed and which should be hidden, and the only thing the main program needs to know, is to which buffer to use while generating the next image. The "buff" variable in our example will always be the hidden buffer with which we should be working.

We have also introduced the SETBLACK command in our last example. SETBLACK allows you to keep the screen "black" during a load process or for any other reason. SETBLACK 1 means "turn black on" or "set the display to black". SETBLACK 1 only modifies the current displayed palette and will not affect the actual image or the palette saved with the buffer. SETBLACK 0 will restore the image display. SETBLACK can also be used any time it may be useful to keep the display off while making adjustments to the current displayed image.

As an exercise, go back to the FAST memory page-flipping example at the end of chapter 3, and modify it to use this "toggling variable" approach to double buffering. I think you will be able to see that it makes the example simpler and more generalized.

Effects

Fades

Fading can be a simple but effective technique for dressing up your sequences. Here is our basic slideshow using a fade out/fade in effect at every picture change:

```

      REM a disk based slideshow
*      FIRST=1
      SETBLACK 1
      BUFF=1
10     LOAD BUFF,":pictures/flower.pic"
      GOSUB 20
      LOAD BUFF,":pictures/tree.pic"
      GOSUB 20
      LOAD BUFF,":pictures/rock.pic"
      GOSUB 20
      LOAD BUFF,":pictures/fish.pic"
      GOSUB 20
      GOTO 10

*20    IF 1=FIRST
*      FIRST=0
*      ELSE
*      FADE 0,3-BUFF,0
*      ENDIF
      DISPLAY BUFF
*      FADE 1,BUFF,0:REM a quick fade in
      BUFF=3-BUFF
      PAUSE 20
      RETURN
```

We will use the asterisk to indicate which lines have been added since the previous example. Do not actually enter the asterisk in your script this way.

This is also an example of the use of the IF/ELSE/ENDIF statements. For more information on these, see the previous section on program flow. Note that most of the additions occur in the subroutine at line 20. By adding our effects here, they will affect all of the picture changes.

We could be more selective if we wished, by adding a second subroutine that does not include our latest additions:

```

        REM a disk based slideshow
        SETBLACK 1
        BUFF=1
10      LOAD BUFF,":pictures/flower.pic"
        GOSUB 20
        LOAD BUFF,":pictures/tree.pic"
*      GOSUB 30
        LOAD BUFF,":pictures/rock.pic"
        GOSUB 20
        LOAD BUFF,":pictures/fish.pic"
        GOSUB 20
        GOTO 10

REM no fade routine
20      DISPLAY BUFF
        BUFF = 3-BUFF
        PAUSE 20
        RETURN

* REM fade routine
* 30      FADE 0,BUFF,0
*      DISPLAY BUFF
*      FADE 1,BUFF,0:REM a quick fade back in
*      BUFF=3-BUFF
*      PAUSE 20
*      RETURN
```

Wipes

The WIPE command can be used to effect several types of wipes. Different types of wipes can also be done with other commands.

Characteristics of Wipes

Most wipes work by slowly (relatively) modifying an old picture with data from a new picture, until the old picture has been completely replaced with the new picture. In this respect, a wipe will replace whatever image is in the screen buffer with the new image being "wiped" in. A wipe does not effect a "display switch" from one screen buffer

to another, but effects a "copy" of the data from one buffer to another. The DISSOLVE command also has this effect.

The WIPE command allows you to "copy" the image information from one buffer to another, as a single wiping pass across the screen. The wipe command will allow you to wipe from top to bottom, bottom to top, left to right, and right to left. The WIPE command can be used to wipe the entire screen, or any smaller rectangular portion of it desired.

Note that one limitation of wipes of this sort, is that BOTH the old and new pictures must be of the same resolution, and if they do not use the same color palette, the transition period during the wipe will show portions of both pictures with the palette for the old picture. Once the wipe is completed, the palette is updated with the new palette. This can cause short periods where some of the new picture is being displayed with the wrong palette.

The DISSOLVE command, and wipes done using the BLIT command have similar palette limitations. The most effective wipe effects will be between pictures of the same resolution with the same palette.

A simple top to bottom wipe effect can be added to the transitions in our slideshow like this:

```

        REM a disk based slideshow
        SETBLACK 1
        BUFF=1
10      LOAD BUFF,":pictures/flower.pic"
        GOSUB 20
        LOAD BUFF,":pictures/tree.pic"
*       GOSUB 40
        LOAD BUFF,":pictures/rock.pic"
        GOSUB 30
        LOAD BUFF,":pictures/fish.pic"
        GOSUB 20
        GOTO 10

REM no effects routine
20      DISPLAY BUFF
        BUFF = 3-BUFF
        PAUSE 20
        RETURN

REM fade routine
30      FADE 0,BUFF,0
        DISPLAY BUFF
        FADE 1,BUFF,0:REM a quick fade back in
        BUFF=3-BUFF
        PAUSE 20
        RETURN

* REM wipe routine
* 40    WIPE BUFF,0,0,0,0,-1,-1,80,0
*       PAUSE 20
*       RETURN
```

The Director will allow you to load several IFF files into memory at once for quick access. Each loaded file is called a "picture buffer", (sometimes referred to simply as "buffer" in the Command Reference chapter) and is referenced by number. You can have up to 30 picture buffers at one time, numbered 1-30, unless you do a BUFFERS command to configure space for a larger number of buffers. With the BUFFERS command, you can have ultimately as many buffers as you could possibly ever fit in your Amiga's memory. A picture buffer may reside in either CHIP or FAST memory, though only CHIP memory buffers can be displayed on the screen directly. When a command in the Command Reference chapter

specifies a buffer, assume the buffer is restricted to CHIP buffers unless otherwise noted. Only the COPY command can be used with FAST buffers.

Dissolve

The DISSOLVE command is similar to a wipe, except the image is transferred as a random scattering of points, rather than a scan across the screen.

Here is a dissolve example:

```
LOAD 1,":pictures/blank.pic"  
LOAD 2,":pictures/pic1.pic"  
DISSOLVE 2,0,0,0,0,-1,-1,3000  
PAUSE 100
```

The first parameter in the DISSOLVE command specifies the buffer to dissolve from, followed by the upper left X and Y locations of both the rectangle to dissolve from, and the rectangle to dissolve to. The two -1's specify the default width and height which is the full screen. The last parameter specifies how many pixels to transfer at a time. The more pixels, the faster the transfer, theoretically. As the number gets larger, slight hesitations between transfers become apparent. The best dissolve speeds are probably in the 500-5000 range for large area dissolves, and very small area dissolves can work effectively with speeds all the way down to 1. Generally, the smaller the display area being dissolved, the faster the dissolve can take place, the speed parameter is relative to the number of pixels in the area specified. The more pixels, the longer the dissolve will take. A full screen hi-res dissolve is fairly slow no matter what parameters are given. Conversely, a small area dissolve can happen extremely fast.

The dissolve can be used for a variety of effects. You could effect a transporter "beam-in" effect by dissolving people into an image of a spaceship transporter room. If the object dissolving in contains the correct background imagery within its rectangle, during the dissolve, ONLY the area within the outline of the person will seem to dissolve.

Stencil

The STENCIL command allows you to use one screen buffer as a stencil during the copy process of another buffer to the screen. STENCIL can be used for specialized wipes, or other effects. STENCIL specifies the buffer you want to copy to the current display buffer, and the buffer you want to use as a stencil. These buffers can be the same buffer if desired.

STENCIL will treat all portions of the stencil buffer that has a color number of non-zero, as a signal to copy corresponding portions of the specified picture buffer to the current screen. If the maskpolarity flag is 0, then all portions of the stencil buffer that has a color number of zero signify pass through.

As an example, if you have an image, let's say "object.pic" that is all color-0 (black let's say) except for an object in the middle of the screen, and you do the following:

```
LOAD 1,":pictures/background.pic"  
LOAD 2,":pictures/object.pic"  
STENCIL 2,2,1  
PAUSE 100
```

What you will see is the image "background.pic" with the object perfectly cut out and deposited into the background, in perfect registration with the position it occupies in the original "object.pic".

In this example, the image being copied and the image used as a stencil are the same image. In another example:

```
LOAD 1,":pictures/blank.pic"  
LOAD 2,":pictures/object.pic"  
LOAD 3,":pictures/background.pic"  
STENCIL 3,2,1  
PAUSE 100
```

Here, the picture buffers are different. What will result is an image the exact shape of "object.pic", but of data given in the picture "background.pic". It is as if you took a cookie cutter the shape of "object.pic" and used it to cut out a "cookie" from "background.pic". The resultant cookie is what you will see at the end of this example.

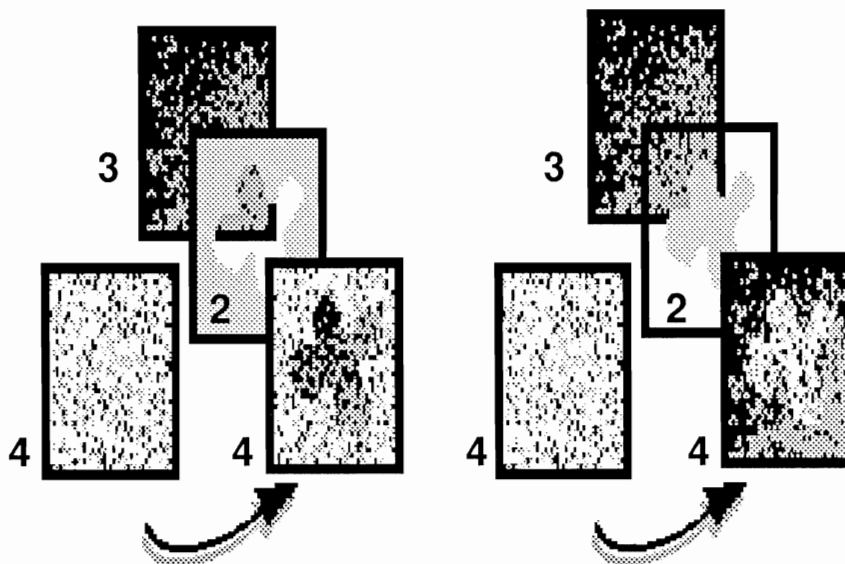
If the last parameter in the STENCIL above were 0, then what you would see, is "background.pic" with an "object.pic" shaped hole cut out of the middle of it.

The STENCIL command can be used to generate custom wipes, by continually modifying the mask image with a wipe pattern while performing successive STENCIL commands.

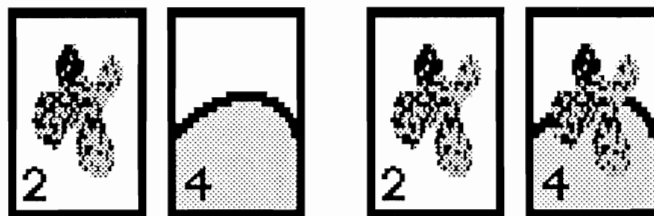
The illustration on the following page may help to make the use of STENCIL clearer.

STENCIL 3, 2, 1

STENCIL 3, 2, 0



STENCIL 2, 2, 1



BEFORE

AFTER

For these examples, buffer 4 is always the displayed buffer. Buffer 4 could represent the hidden buffer in a double-buffered sequence if double-buffering was being done at the same time.

In the example in the upper left, buffer 3 is being copied to the displayed buffer, buffer 4. Buffer 2 is being used as a stencil. After the STENCIL 3,2,1 command, buffer 4 contains portions of buffer 3, where buffer 2 correspondingly was not color 0.

In the example in the upper right, the STENCIL 3,2,0 command does the same thing, except buffer 4 then contains portions of buffer 3 where buffer 2 correspondingly WAS color 0.

In the example at the bottom, STENCIL 2,2,1 uses buffer 2 as BOTH the stencil buffer and the buffer to copy from. This results in everything on buffer 2 that is not color 0 to be copied to buffer 4.

BLIT

The BLIT command allows you to move rectangles of image information from one screen buffer to another. BLIT is taken from the word "blitter" which is derived from "bitblt" which is short for Bit BLock Transfer. A blitter is a hardware device that can move image information very fast with a minimum of main computer processor support. The main processor can tell the blitter to do an operation. In a multitasking system such as the Amiga, while the blitter is performing the instruction, the main processor is free to work on other things.

The Amiga's blitter provides the ability to move image information around at high speed. The BLIT command ties directly into the blitter (as do many of the Director's commands).

A typical BLIT command looks like this:

BLIT 2,79,62,220,50,30,30

The BLIT command parameters are identical to the first 7 parameters in the DISSOLVE and WIPE commands. The first parameter is the screen buffer to blit from, followed by the X and Y location of the upper left corner of the rectangle to blit, followed by the X and Y location of the upper left corner of where the rectangle should go after it has been BLIT, and finally, the width and height of the rectangle itself.

The BLIT command is fast enough to do partial screen page-flipping.

This may be a common technique when using the Director. By building a screen with a paint package that contains a series of "frames" of the portion of the screen that is to change by "flipping", you can then use the BLIT command to move them into position on your displayed screen.

If transparency has been turned on with the TRANSPARENT 1 command, use of the BLIT command will result in transfer of only the areas within the selected rectangle that are not color 0 (the background color). This will allow you to place a non-rectangular shaped object on top of a backdrop image. We have discovered that in many cases, it is still more effective to do a non-transparent BLIT because it has the effect of erasing over previously BLITed imagery that might be visible otherwise.

A BLIT of image data is not required to be contained entirely on screen. It may be desirable to begin to BLIT an image from completely off screen and then slowly move it on screen. The BLIT command provides full edge-of-screen clipping so that this is possible.

The BLIT command can be used to generate customized wipes, by copying portions of one screen to another a bit at a time in some prescribed pattern. A checkerboard wipe would be one example of a BLIT oriented custom wipe.

In order to make it easier to determine what the parameters for the BLIT command should be in a given situation, a Director script called "blitutil" and the resultant .film file are included on the Director disk. This utility allows you to load the images involved, and visually specify the rectangular areas. The utility will then display the required BLIT command parameters to accomplish the specified move. There is more information on Blitutil in chapter 6.

A detailed example of an animation that uses the BLIT command is outlined in chapter 5.

Adjusting Speed

At times it may be necessary to pause at increments smaller than tenths of a second. In particular, when doing page-flipping or quick effects, a tenth second pause can be too long. By using the SPEED command, you can adjust this increment. The speed command serves as a multiplier for all PAUSE commands, and its default value is 5. This means that if you do a SPEED 10, then all of your pauses will take twice as long. Or if you do a SPEED 1, then all of your pauses will take one-fifth as long. With SPEED 1, pause commands all work in one-fiftieth of a second

counts. You can, if you desire, use the SPEED command to modify the timing of large sections of your scripts where you are presently using PAUSE commands. By changing the SPEED command, you can speed up or slow down your entire presentation. The command SPEED 0 will have the same effect as removing all of your PAUSE commands.

Remember that if you decide to adjust the pause speed with the SPEED command during your scripts, and you have other portions of your script which were designed using the default SPEED value, that you should restore SPEED to the default value (with SPEED 5 or SPEED -1) after you have finished a section that required a speed adjustment.

Text and Fonts

Text

The Director will allow you to generate text slides, or annotate your image slides with text generated by the Director. One advantage to using the Director's built in text generation, is that you don't have to build and save lots of IFF images with text on them, which could take up a lot of disk and computer memory space. By generating text directly with the Director, the actual storage of the text takes up very little space. You could sequence hundreds of textual slide images on a 512k system, without accessing the disk during the presentation.

Here is an example that displays text on the screen:

```
LOAD 1,":pictures/blank.pic"  
PEN 1,15  
MOVE 10,10  
TEXT "Here is some text"  
MOVE 10,40  
TEXT "The number is:";?100  
PAUSE 50  
END
```

We first load a blank picture to use as a background. This could also be a picture that has border designs to be used with our text. We then set the pen color for the foreground pen, to color 15. We use the MOVE command to move our current graphics position to 10X,10Y. When we do the TEXT command, our text will start at this X,Y position. And finally, we have an example of mixing text output with numeric data

output in a combination text statement.

Here is how we can center our text automatically:

```
LOAD 1,":pictures/blank.pic"  
CENTER 1  
PEN 1,15  
MOVE 10,10  
TEXT "Here is some text"  
MOVE 10,40  
TEXT "The number is: ";?100  
PAUSE 50  
END
```

Once you turn auto-centering on with the CENTER command, the X coordinate of the MOVE commands are disregarded on TEXT commands, and the text is centered between the default margins (edges of screen) or whatever margins have been set up with the MARGINS command. You can change the margin settings with the MARGINS command, and then output more text with the TEXT commands to be centered between different areas on the screen.

You can use the STYLE command to select underline, bold, or italic versions of your selected font. STYLE can also be used to adjust the inter character spacing of your text.

Foreground and Background

Normally when text is displayed on the screen by the Director, the text (foreground) is rendered with the color from pen 1, and the box space the text occupies (background) is rendered with the color from pen 0. Sometimes it may be desirable to display only the foreground of the text, and not displaying the background. This can be done by using the DRAWMODE 0 command. With this command, you can display text over graphics images where only the text itself is rendered, and there is no little "rectangle" cut out of the background around each letter. You can return the Director to the default mode of displaying both foreground and background with the command DRAWMODE 1.

Fonts

Multiple fonts can be selected from the Director for quick access. Fonts can be preloaded so that there is no disk activity required when you change from one font to the next which might otherwise slow down displays.

Configuring for Fonts

In order to load a font for use, a fonts: directory must be available. To use the standard Amiga supplied fonts, you can either copy the fonts directory to your working disk, or reference them from another drive. The Amiga CLI command ASSIGN is normally used to tell the system which directory to use when looking for available fonts. If you have a fonts: directory on the disk you booted from, and the fonts you want to use are there, you shouldn't need to do anything except to just use them from the director. If you have fonts on another disk you want to use, you can do the CLI command:

Assign fonts: otherdisk:fonts

From then on, any time any program needs to load a font, it will look on this disk, in the specified directory.

In the event you wish to package a disk to be distributed that includes fonts that you use in a Director script, keep in mind that many fonts are not public domain, and can't be legally distributed with your sequence. The Amiga system fonts, as an example, aren't public domain (at the time of this writing). If you have some fonts that you know are public domain, then you can legally include them on your disks for distribution.

It may also be desirable to provide a disk that is completely filled with animation files, and contains no system files and therefore will not boot. At the same time, you may want to place your public domain fonts on this disk, and need a friendly way to set up your animation so that it will automatically re-assign the system fonts: location to your disk. To do that, you can do a:

EXECUTE v,"Assign fonts: mydisk:fonts"

before you ever actually do a LOADFONT from within your script. This will cause the system to automatically setup the correct fonts: location for your script. When your script finishes, the fonts will still be assigned to your disk, which may cause some confusion to programs run later. You can handle that problem, by putting this up at the beginning of your program:

```
EXECUTE v,"Assign temp: fonts:"  
EXECUTE v,"Assign fonts: mydisk:fonts"
```

And then just after your LOADFONTs, do:

```
EXECUTE v,"Assign fonts: temp:"
```

Which will restore the font directory back to where it was when the sequence was started.

Determining Which Fonts Are Available

If you don't know what fonts are available to you from the Director, from the CLI you can do a:

```
cd fonts:  
dir
```

This will give you a list of all of the font names available. Each font will have its name as a file with a .font on the end. If you then use the DIR command to find out what is in the corresponding directory:

```
dir diamond
```

Where there exists a diamond.font in the list you got from the previous example, you will get a list of all of the available point sizes for that font.

Using Fonts from Scripts

To load some fonts with the Director, try something like this:

```
LOAD 1,"blank.pic"  
PEN 1,15  
LOADFONT 1,8,"topaz.font"  
LOADFONT 2,12,"ruby.font"  
LOADFONT 3,9,"diamond.font"  
SETFONT 1  
MOVE 20,20  
TEXT "This is in topaz 8"  
SETFONT 2  
MOVE 20,40  
TEXT "This is in ruby 12"  
SETFONT 3  
MOVE 20,60  
TEXT "This is in special 9"  
PAUSE 50  
END
```

The first parameter of the `LOADFONT` command specifies the font buffer to load the font into. Font buffers use a separate set of buffers than the screen buffers, so you do not have to worry about conflicts with those. The second parameter is the point size of the font, and the last parameter is the full font name.

The `SETFONT` command then selects which font, by font buffer number, that you want to use at any given time. Since the fonts are then already loaded, you can `SETFONT` as much as you like, and it won't cause disk accesses. Thus you can very quickly change fonts for fast displays where you otherwise don't want to wait for the font to load from disk.

You can free up the memory space being used by such resident fonts with the `FREEFONT` command:

```
FREEFONT 2
```

Will free the memory for the font that was loaded into font buffer 2.

More Double Buffering

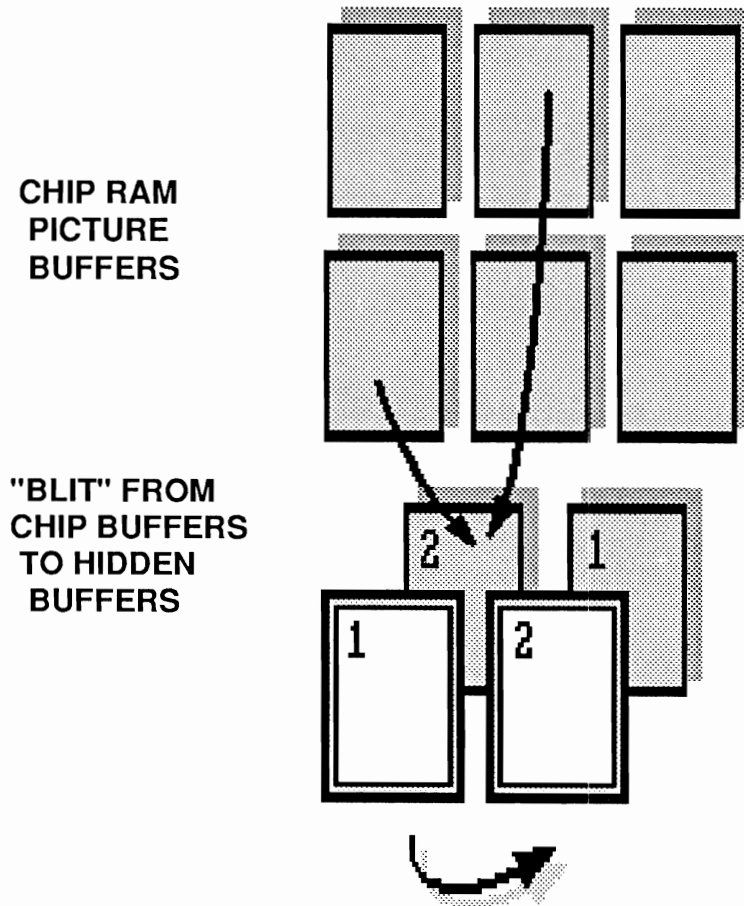
Double buffering is a technique used to allow graphics effects that might upset the screen display if done to the current screen directly and clean them up by using a second screen buffer to do the actual screen commands, and then when done, making that buffer the new current screen. This can be used to eliminate undesirable blitter effects, or make it look as if several graphics commands have actually occurred at exactly the same instant.

An important command to aid in double buffering is the BLITDEST command. Normally, when you do a BLIT or any drawing command, the displayed screen buffer is where the new image is created. Sometimes it is useful to generate the next image on a non-displayed screen, to be displayed when all image generation is complete. To do this, you can redirect all BLIT and drawing commands (including TEXT) to a non-displayed buffer simply by using the command BLITDEST #, where # is the number of the buffer commands should be redirected to. BLITDEST -1 will return the destination of the drawing commands to the currently selected screen buffer, and the DISPLAY command will reset this to be the displayed screen buffer as well.

BLITDEST can be very useful when double buffering.

Let's look at the illustration at the next page as an example.

DOUBLE BUFFERING



If we want to do several BLIT modifications to our screen, we may not want to do them one at a time. While they may be fast, if there are enough of them, they may not appear to happen exactly at the same time if we are not double buffering. In the illustration, the upper boxes represent buffers that contain imagery that we want to BLIT to the screen. We can do multiple BLITs to our hidden buffer, then display it which causes our previously displayed buffer to become our new hidden buffer. Here's a program that double buffers multiple BLIT commands:

```

                                SETBLACK 1:REM come up black
                                LOAD 3,":pictures/tree.pic"
                                LOAD 4,":pictures/other.pic"
                                NEW 1,3
                                NEW 2,3
                                DISPLAY 2
                                SETBLACK 0:REM turn on screen
                                B=1
10                                BLITDEST B
                                CLEAR
                                BLIT 3,0,0,20,38,20,20
                                BLIT 4,15,90,280,110,30,30
                                GOSUB 99:REM do display now
                                GOTO 10

99                                DISPLAY B
                                B=3-B
                                BLITDEST B
                                PAUSE 1
                                RETURN

```

In this example, as in the illustration, screens 1 and 2 are used to do the double-buffering. A variable, "B" is used to keep track of the non-displayed buffer.

The BLITDEST command redirects the results of most of the drawing and BLITing commands to a buffer other than the currently displayed one. BLITDEST remains in effect until the next DISPLAY command, at which time the results of the commands are directed back to the buffer being DISPLAYed.

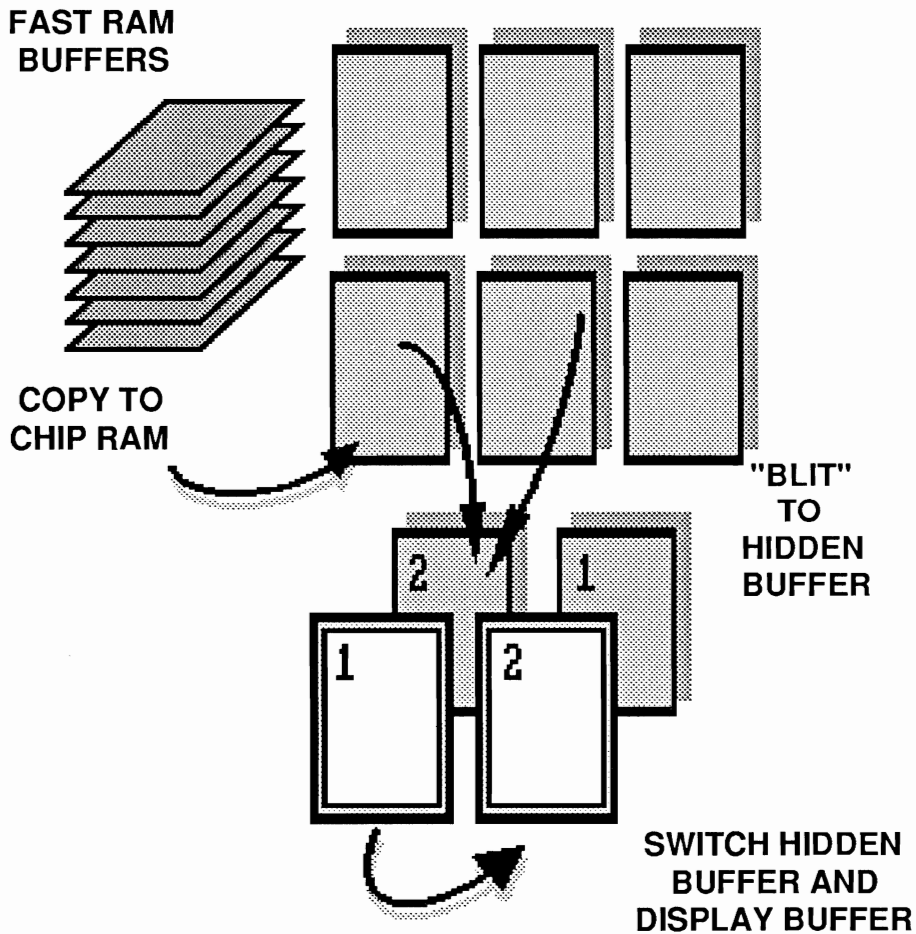
By using the BLITDEST B command, BLIT's and associated other drawing commands are used to construct the correct picture on the non-displayed buffer. Then, the subroutine at 99 is called, which will then make the non-displayed buffer the displayed buffer, convert B to the other buffer number (every time GOSUB 99 is executed, B will flip from 2 to 1 and

then back to 2 etc.). BLITDEST B is then used to cause all commands (in this case the BLIT command) to use the non-displayed buffer for their output instead of the displayed buffer.

Any collection of graphics commands can be done in place of the CLEAR and BLIT etc., if it is important to double-buffer for image stability. To see what would happen if you tried to do this effect without double-buffering, take out the BLITDEST B and the GOSUB 99 commands. In some cases, you may find double-buffering is not necessary, and that even though you are BLIT'ing imagery directly to the displayed screen, it does not produce any objectionable side effects. Other times, only double-buffering will provide the clean transitions you need. Experimentation and experience should help you in deciding what you should do for a given situation.

The illustration on the next page adds fast ram images to our scenario.

DOUBLE BUFFERING USING A COMBINATION OF CHIP RAM AND FAST RAM



Here, we can use COPY to move fast ram buffers into chip ram, and then use BLIT to move the imagery to the hidden buffer being used to construct the next image.

Drawing Commands

With the Director you can also draw graphics directly to the screen. The commands POINT, MOVE, DRAW, CIRCLE, ELLIPSE, RECT, FILL, and GETPEN will allow you to draw your own images or add other effects to your presentations.

POINT allows you to display single points, MOVE allows you to move an invisible marker to some point on screen, DRAW will draw a line from the invisible marker to a specified point, CIRCLE and ELLIPSE will draw circles and ellipses, RECT will draw a solid filled rectangle, and FILL will solid fill in an enclosed area on screen. GETPEN allows you to examine specific pixels on the screen one at a time. See the Command Reference chapter for specifics on each of these commands.

The PEN command is used to set the "pen" color register for such drawn objects to be drawn with. Pen 1 is the normally used pen for the lines and circles to be drawn with. Pen 0 is the background pen, and is primarily only used for text backgrounds. Pen 2 is the outline pen, and is used by the FILL command. The RECT command will draw a box that has its edge outlined with the specified outline color if the outline pen is non-0.

Color

PEN specifies which pen to be used, and assigns a specific color register to be used. This will pick a number out of the current palette. The palette size depends on the screen resolution of the screen you are drawing on, and is equivalent to the paint program palette used to generate the background image.

The COLOR command can be used to modify specific colors in the palette. COLOR can be used to set any palette color to any of the 4096 available Amiga colors. Since the Director keeps a copy of the color palette separate from the actual visible screen palette, you can choose to just modify the visible screen palette so that later you can restore the original palette with the PALETTE command.

On the other hand, if you want to make a palette color change that is

remembered for the rest of the Director sequence, you can use COLOR with permanence set to 1 to cause this to happen.

If you want to do more sophisticated color table manipulations, you can read the entire color palette into the array using the GETMAP command, and write the color palette from one constructed or read in and modified kept in the array with the SETMAP command.

Shaking the Bugs Out

As you experiment with more sophisticated sequences, there may be times where a script file seems to be doing something different than you told it to. Usually, this is because you forgot to do something, or may have typed something in wrong.

In many instances, if the Director finds that something you have told it to do doesn't make sense, the Director will give up and report an error message to you. The error messages are designed to help you to decide what is wrong so you can correct the problem.

Sometimes there may be errors in your Director script that are perfectly valid Director commands, but one or more of them are either not doing what you thought they would do, or may have mis-typed them in a way so the Director still thought them valid.

Using the trace mode, you can cause the Director to output a message for each command that it executes, allowing you to monitor what is actually going on. This can help you to follow the Director as the sequence is performed, to see exactly where it decided to take a turn that is not what you were expecting.

When you invoke the Director from the CLI, you can tell it to run the entire script with the trace mode on, by specifying the command:

```
director filename opt t
```

or

```
director -t filename
```

You can only run traces on script files, not on .film files. Attempts to run traces on .film files will result in no action.

In addition, you can also decide to modify your script, and put the

commands TRON and TROFF to turn the trace mode on and off across portions of your script that may be suspect. Using these commands, you won't have to wait while the trace information is output to the CLI through long portions of a sequence which are working correct, but can frame the problem areas so only they are being monitored.

At times, it may be inconvenient to watch the trace information as it is output to the CLI during a sequence. If this is the case, you can redirect the trace information to a text file with a command like:

```
director >trace.text filename opt t
```

This technique will work whether or not you use the "opt t" or the TRON and TROFF commands to generate the trace information. After you exit the Director, you can then look at the "trace.text" file to see what happened.

Trace information will display the command name, followed by a list of the parameters actually computed separated by commas. Text information will show up as a single text string. In the event an error occurs during a parameter evaluation, you should see that error message occurring in the text at the exact location within the command parameter string where the error was detected.

Variables do not show up with their actual variable name in trace output. Variable names are converted to an internal numeric representation and are not retained throughout the execution of Director sequences. For this reason, variable assignments in trace statements will look something like:

```
[VAR03] = 2
```

By comparing the trace output with the actual Director script, you can determine exactly to which statement such variable assignments refer.

Freeing Memory

If during a Director sequence, you would like to free up memory that is being used by various screen buffers or anim buffers, you can use the FREE command to release that memory selectively for reuse by later portions of the same Director sequence.

Memory Monitoring

You can determine just how much memory is available to your Director sequence during the sequence, with the MEMORY command. This command will set three variables to the values of your total memory, and the available chip and fast memory. By checking these values, you can determine if you have enough memory to perform certain functions, or can decide if an alternative low-memory sequence should be performed instead.

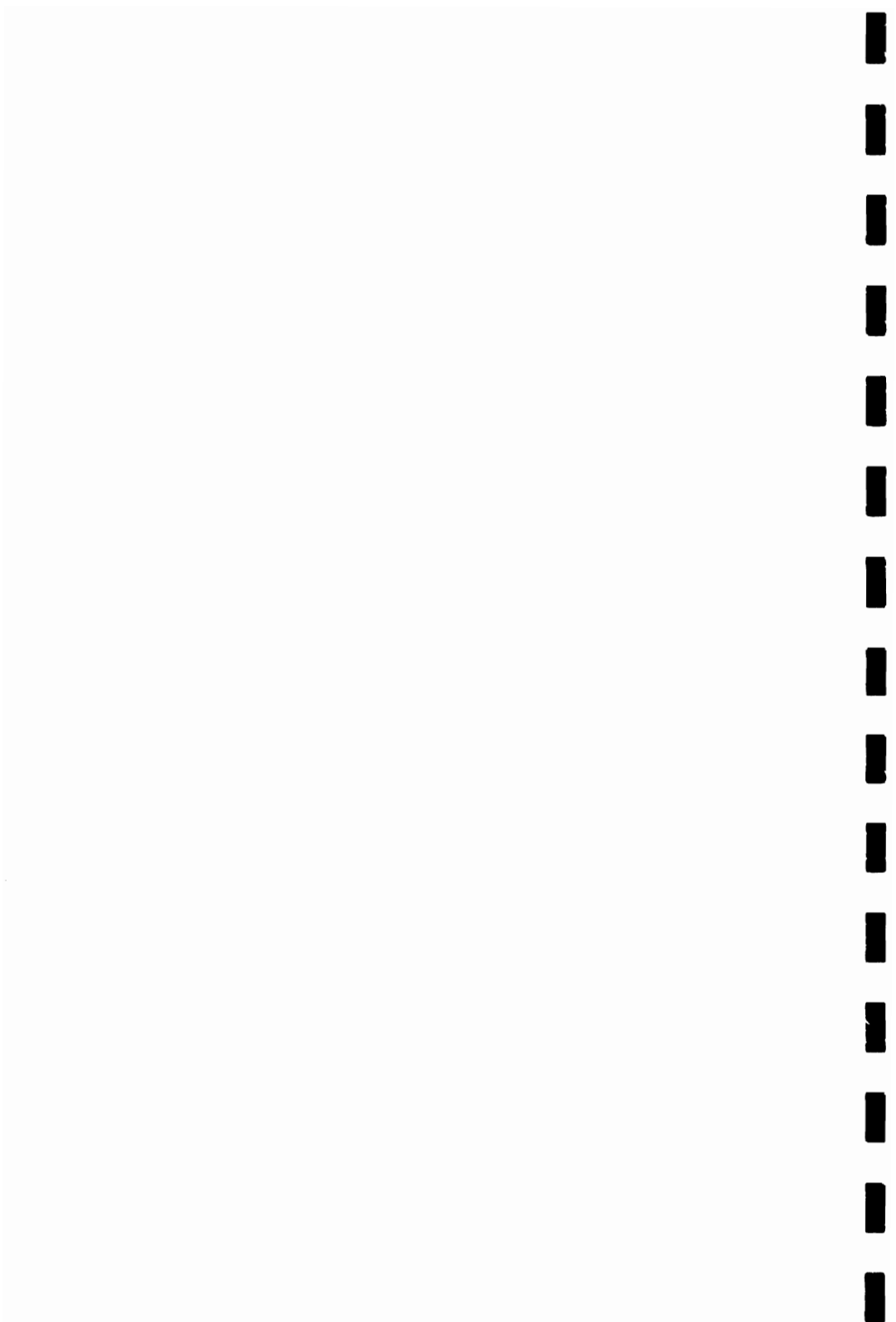
Use the memory command like this:

```
MEMORY all,chip,fast
PRINT "You have ";all;" bytes total"
PRINT "with ";chip;" bytes of chip memory available"
PRINT "and ";fast;" bytes of fast memory available"
```

Instead of the PRINT statements, you can test the three variables to decide if you have enough memory to load more frames, or to decide to do things in different ways depending on available memory.

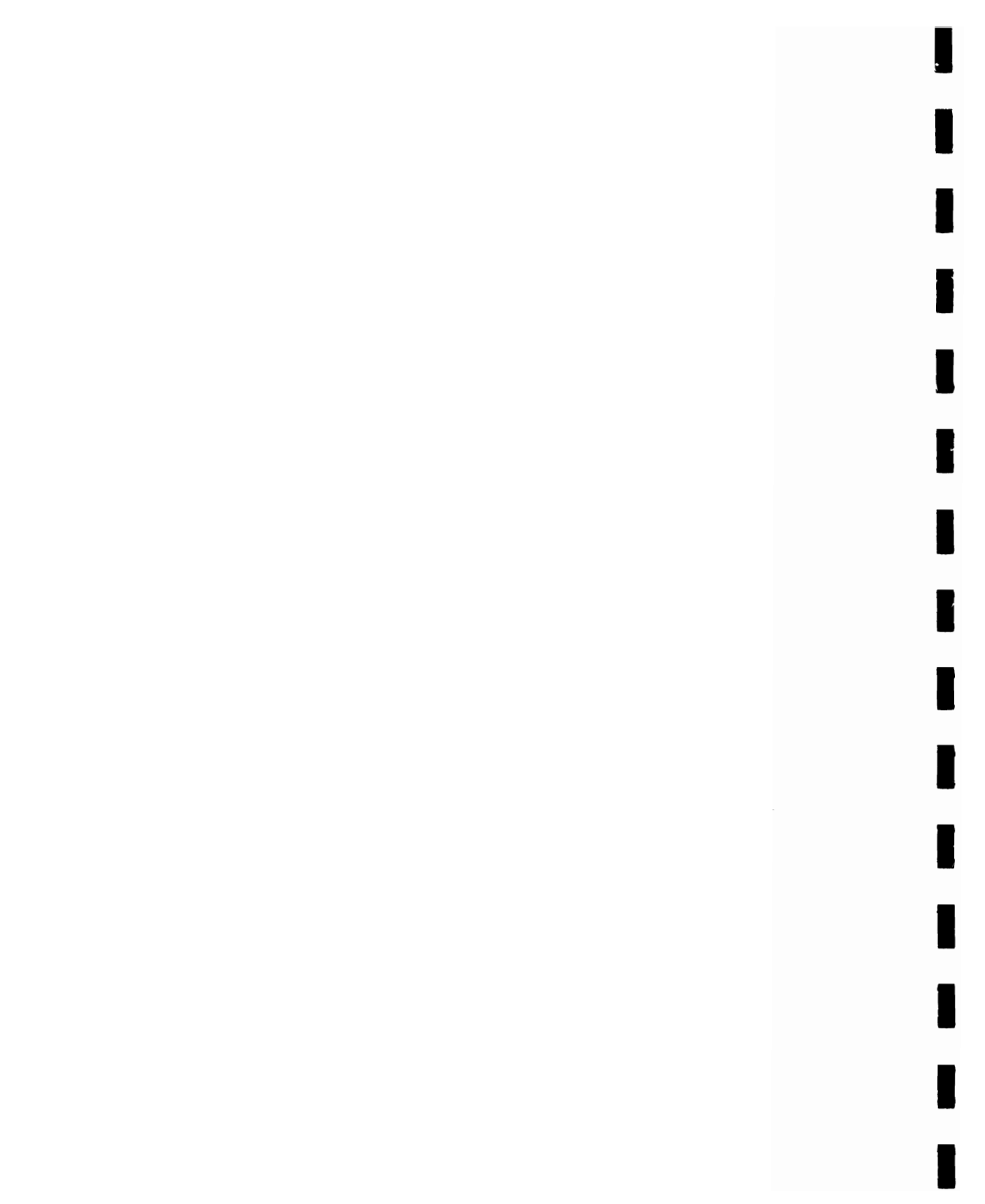
Ending Sequences

Besides mouse abort, you can explicitly terminate a Director sequence with the END command.



5

PUTTING THE PIECES TOGETHER



5. Putting the Pieces Together

Programming With the Director

You should have a reasonable understanding of how the basic Director commands work from the previous sections. To see how these can actually go together we will take you step by step through a display and animation script. If you are new to the idea of programming, this will help you see how to put commands together to create an effective computer film.

One approach to programming your computer film is to proceed in small steps. Each step of the way, you can verify that your additions to the program work as you intended. In this way, complex programs can be built in easy to understand stages. As you look through some of the more complex scripts included on the Director disk, remember that they were built up this way. Don't be intimidated by long pages of commands and numbers. Each of these examples began as a small, simple group of statements, and evolved in easy, logical steps.

For this example we will use three picture files contained in the TUTORIAL directory on your disk. They are:

```
FRAME  
IMAGES1  
IMAGES2
```

Look at them before starting, so you understand the graphic approach being used. One picture file is the main background for the display and animation we will create. The other two picture files are screens packed with useful images and sequences of frames for animations. All were created with a paint package, or a digitizer. To look at them, use your paint package, or one of the public domain IFF viewer utilities like "seeilbm", "show", "ushow" or "view".

Since these pictures are stored in a directory, you must include the name of the directory when you call for them from the disk. Use your text editor (see "Getting Started") to open a new file for the script you are about to write. Name it Testscript.

Let's begin:

Using your editor, type into your file:

```
LOAD 1,":tutorial/frame"
```

If you are new to programming, you have just written your first program. To run it, from the CLI, type:

```
director testscript
```

You saw the picture you called for flash briefly on the screen before the program returned you to the CLI. Not a very impressive program yet, but it is doing what you expect it to do. This may seem ridiculously obvious, but it is very important. Let's expand the program into a better foundation from which to proceed. Add these lines:

```
LOAD 1,"tutorial/frame"  
*   LOAD 3,"tutorial/images1"  
*   LOAD 4,"tutorial/images2"
```

Save the revised script.

Now run it to see what has happened:

```
director testscript
```

The picture stays on screen longer this time. It remains visible while the other pictures are being loaded from the disk into the chip ram buffers. We don't see those pictures because there has been no command to display them. Only the picture in the first LOAD command is displayed. Let's continue adding to the script by calling for a portion of the picture in buffer 4 to be added to the displayed buffer 1 picture:

```
LOAD 1,"tutorial/frame"  
LOAD 3,"tutorial/images1"  
LOAD 4,"tutorial/images2"  
*   BLIT 4,2,60,5,4,264,136  
*   PAUSE 50  
*   END
```

Review the Command Reference section of the manual if you are unsure of the meaning of new commands. Had you built "images1" yourself, you would have used BLITUTIL or the co-ordinates feature in your paint package to keep a record of the location and dimension of each image,

its origin and destination. Review the use of BLITUTIL and BLIT if you don't understand the numbers you just entered with BLIT. Save the revised script and run it:

`director testscript`

The display in buffer 1 is beginning to take shape. Note that the picture in buffer 1 has been permanently changed by this addition. If you were to display another picture buffer at this point, buffer 1 could not be seen, but would continue to hold this image and could be redisplayed at any time.

BLIT is a powerful feature of the Director. You can utilize available memory much more economically by being able to pack many images onto each screen you save as a picture file to your disk. In writing your script, BLIT allows you to select any part of that picture file to be displayed to the screen. If you look at the picture file, "images2", you see the image of the young chemist and other images obviously intended to be used with the picture, "frame". We have used BLIT to bring only the young chemist into the displayed picture without needing to change the whole picture. Note that BLIT does not change "images2" in any way. The image of the boy is still there.

Let's continue by adding some animation. Working with relatively small areas of the screen at a time, it is possible to use BLIT to animate directly onto the displayed screen without double buffering. If we were animating a large part of the screen, the double buffer technique would be useful to keep the display absolutely clean and smooth.

For our first addition, let's use the six atom pictures on "images1". We will animate them in the displayed frame before we put the boy's picture there. You would use your record of locations and dimensions for these images had you made them yourself.

Add these lines:

```
        LOAD 1,"tutorial/frame"
        LOAD 3,"tutorial/images1"
        LOAD 4,"tutorial/images2"
*       BLIT 3,1,47,15,35,84,74:PAUSE 1
*       BLIT 3,88,47,15,35,84,74:PAUSE 1
*       BLIT 3,175,47,15,35,84,74:PAUSE 1
*       BLIT 3,1,124,15,35,84,74:PAUSE 1
*       BLIT 3,88,124,15,35,84,74:PAUSE 1
*       BLIT 3,175,124,15,35,84,74:PAUSE 1
        BLIT 4,2,60,5,4,264,136
        PAUSE 50
        END
```

Remember that the colon (:) in these lines is a way of putting more than one command on the same line. Save the new script and run it:

```
director testscript
```

It's getting better. The atom is in the right spot, and moves properly and cleanly. This is "page flipping" animation. Now let's make it continue to run more than just one cycle. We will add the concept of "looping" using the FOR and NEXT commands to do eight complete cycles of the atom animation before moving on to the next group of commands. Add these lines:

```
        LOAD 1,"tutorial/frame"
        LOAD 3,"tutorial/images1"
        LOAD 4,"tutorial/images2"
*       FOR a=1 to 8
            BLIT 3,1,47,15,35,84,74:PAUSE 1
            BLIT 3,88,47,15,35,84,74:PAUSE 1
            BLIT 3,175,47,15,35,84,74:PAUSE 1
            BLIT 3,1,124,15,35,84,74:PAUSE 1
            BLIT 3,88,124,15,35,84,74:PAUSE 1
            BLIT 3,175,124,15,35,84,74:PAUSE 1
*       NEXT
        BLIT 4,2,60,5,4,264,136
        PAUSE 50
        END
```

Save the new script and run it:

```
director testscript
```

That worked out well, (if not, check your typing). Remember, these numbers in the BLIT command simply specify where the image is, how big it is, and where you want to put it. We could be more specific about defining where to put it by using the BLITDEST command, but in this case, the desired BLIT destination has not changed since the beginning of the program. It remains at the default destination of the displayed screen resulting from the first LOAD command. Let's add another animation to the screen, the little "fission" sequence from "images1". Add these lines:

```
LOAD 1,"tutorial/frame"
LOAD 3,"tutorial/images1"
LOAD 4,"tutorial/images2"
FOR a=1 to 8
  BLIT 3,1,47,15,35,84,74:PAUSE 1
  BLIT 3,88,47,15,35,84,74:PAUSE 1
  BLIT 3,175,47,15,35,84,74:PAUSE 1
  BLIT 3,1,124,15,35,84,74:PAUSE 1
  BLIT 3,88,124,15,35,84,74:PAUSE 1
  BLIT 3,175,124,15,35,84,74:PAUSE 1
NEXT
BLIT 4,2,60,5,4,264,136
* PAUSE 10
* FOR f=1 to 15
*   BLIT 3,1,5,255,150,52,39:PAUSE 1
*   BLIT 3,56,5,255,150,52,39:PAUSE 1
*   BLIT 3,111,5,255,150,52,39:PAUSE 1
*   BLIT 3,166,5,255,150,52,39:PAUSE 1
*   BLIT 3,263,36,255,150,52,39:PAUSE 1
*   BLIT 3,263,77,255,150,52,39:PAUSE 1
*   BLIT 3,263,118,255,150,52,39:PAUSE 1
*   BLIT 3,263,159,255,150,52,39:PAUSE 1
* NEXT
* PAUSE 50
* END
```

Save it and run it:

```
director testscript
```

Everything works well. You will not always be so lucky. Often you will find that your BLIT co-ordinates are off a pixel, or even completely wrong because of a typo. You will inevitably make many mistakes as you work to achieve a particular effect. The benefit of this periodic check of the program at each stage of development becomes evident. You are much more likely to know immediately where a problem lies, because the previous stages all worked well. Corrections are easily made.

Let's begin dressing the program up with some text. Use the TEXT command with the default system font in this example rather than loading in a custom font (see LOADFONT and SETFONT). PEN allows us to use different colors for text, corresponding to the colors in the palette of the displayed screen, in this case. DRAWMODE 0 lets us put text on the screen with no transfer of background color around it. MOVE tells the computer exactly where to put the text. Add these new lines:

```
LOAD 1,"tutorial/frame"
LOAD 3,"tutorial/images1"
LOAD 4,"tutorial/images2"
* DRAWMODE 0:PEN 1,15:MOVE 115,60
* TEXT "LEARN TO CONSTRUCT":MOVE 115,75
* TEXT "NUCLEAR DEVICES IN":MOVE 115,90
* TEXT "YOUR OWN GARAGE..."
FOR a=1 to 8
  BLIT 3,1,47,15,35,84,74:PAUSE 1
  BLIT 3,88,47,15,35,84,74:PAUSE 1
  BLIT 3,175,47,15,35,84,74:PAUSE 1
  BLIT 3,1,124,15,35,84,74:PAUSE 1
  BLIT 3,88,124,15,35,84,74:PAUSE 1
  BLIT 3,175,124,15,35,84,74:PAUSE 1
NEXT
```

```

        BLIT 4,2,60,5,4,264,136
        PAUSE 10
*      MOVE 20,165
*      TEXT "TAME THE ATOM FOR":MOVE 20,180
*      TEXT "PEACEFUL PURPOSES"
        FOR f=1 to 15
            BLIT 3,1,5,255,150,52,39:PAUSE 1
            BLIT 3,56,5,255,150,52,39:PAUSE 1
            BLIT 3,111,5,255,150,52,39:PAUSE 1
            BLIT 3,166,5,255,150,52,39:PAUSE 1
            BLIT 3,263,36,255,150,52,39:PAUSE 1
            BLIT 3,263,77,255,150,52,39:PAUSE 1
            BLIT 3,263,118,255,150,52,39:PAUSE 1
            BLIT 3,263,159,255,150,52,39:PAUSE 1
        NEXT
        PAUSE 50
    END

```

Save the script and run it:

```
director testscript
```

Conditional statements use IF and ENDIF. These two commands allow you to have the program do something when certain conditions exist, and only if those conditions exist. We are going to use them within one of the FOR/NEXT loops to display new text and images while the "fission" animation is running. In the FOR/NEXT loop, the variable (in this case, f) increases with each loop until, in this case, it equals 15. At that point, the program continues beyond the NEXT statement. The variable can be used as a counter, telling the computer to do something when the variable equals five, for example, and do something else when it equals twelve.

The IF line tells the computer what conditions to wait for. The statements that follow the IF line tell the computer what to do when that time comes. ENDIF closes the command. Let's use the CYCLE command in this example. You will have set up the palette in your paint package to cycle only those colors you select. This information is automatically saved with the picture. Now you tell the computer whether to turn the cycling on or off. PALETTE will be used to make sure the colors return to their original positions when the cycle is finished.

Add these lines:

```
LOAD 1,"tutorial/frame"
LOAD 3,"tutorial/images1"
LOAD 4,"tutorial/images2"
DRAWMODE 0:PEN 1,15:MOVE 115,60
TEXT "LEARN TO CONSTRUCT":MOVE 115,75
TEXT "NUCLEAR DEVICES IN":MOVE 115,90
TEXT "YOUR OWN GARAGE..."
FOR a=1 to 8
  BLIT 3,1,47,15,35,84,74:PAUSE 1
  BLIT 3,88,47,15,35,84,74:PAUSE 1
  BLIT 3,175,47,15,35,84,74:PAUSE 1
  BLIT 3,1,124,15,35,84,74:PAUSE 1
  BLIT 3,88,124,15,35,84,74:PAUSE 1
  BLIT 3,175,124,15,35,84,74:PAUSE 1
NEXT
BLIT 4,2,60,5,4,264,136
PAUSE 10
MOVE 20,165

TEXT "TAME THE ATOM FOR":MOVE 20,180
TEXT "PEACEFUL PURPOSES"
FOR f=1 to 15
*   IF f=5:BLIT 4,2,14,13,150,150,38 :ENDIF
*   IF f=7:MOVE 20,165
*   TEXT "ORDINARY VINEGAR-":MOVE 20,180:ENDIF
*   IF f=9:cycle 1:ENDIF
*   IF f=11
*   TEXT "FUEL FOR TOMORROW"
*   ENDEF
*   IF f=13:CYCLE 0:PALETTE 1,0:ENDIF
  BLIT 3,1,5,255,150,52,39:PAUSE 1
  BLIT 3,56,5,255,150,52,39:PAUSE 1
  BLIT 3,111,5,255,150,52,39:PAUSE 1
  BLIT 3,166,5,255,150,52,39:PAUSE 1
  BLIT 3,263,36,255,150,52,39:PAUSE 1
  BLIT 3,263,77,255,150,52,39:PAUSE 1
  BLIT 3,263,118,255,150,52,39:PAUSE 1
  BLIT 3,263,159,255,150,52,39:PAUSE 1
NEXT
PAUSE 50
END
```

Now the display shows a little variety. The new IF/ENDIF lines in the example were written in different forms to show you the latitude you have in structuring your statements. Notice that each IF statement eventually ends with ENDIF. All the conditional statements were put at the beginning of the animation loop so the computer could check them all each time before doing the animation loop. In this way, any small delay always happens at the same place in the sequence and will not be noticed. As you see, the script is getting longer and beginning to look complicated. As you have seen, each step of the way has been an understandable stage. Patient effort, one step at a time, builds an impressive display in the long run.

As you are programming, you may want to use REM to put clarifying comments into the script. These remarks will be ignored by the computer, they are for your use. In the last change we will make to the script, we will add REM remarks to make the script easier to understand when you return to it in the future. The symbol # is the same as REM.

For the final additions, let's polish the presentation a bit. We will add fades at the beginning and end, put a drop shadow title on the screen, and emphasize the vinegar in the flask using a circle. PEN 1 has been white up until now, color 15 on this palette. For the effects we are going to do, we will make use of other colors. We also will use SETBLACK, CIRCLE and FADE. Remember to use MOVE to tell the computer where to put text and circles. Add these lines:

```
*      SETBLACK 1:REM    screen comes up black...
      LOAD 1,"tutorial/frame"
      LOAD 3,"tutorial/images1"
      LOAD 4,"tutorial/images2"
      DRAWMODE 0:PEN 1,15:MOVE 115,60
      TEXT "LEARN TO CONSTRUCT":MOVE 115,75
      TEXT "NUCLEAR DEVICES IN":MOVE 115,90
      TEXT "YOUR OWN GARAGE..."
*      MOVE 185,160:PEN 1,24:REM    dark blue color...
*      TEXT "TEST":MOVE 185,172:TEXT "SCREEN"
*      MOVE 183,158:PEN 1,27:REM    light blue color...
*      TEXT "TEST":MOVE 183,170:TEXT "SCREEN":PEN 1,30
*      FADE 1,1,0:PAUSE 10
```



```

* REM  atom animation begins here...
      FOR a=1 to 8
        BLIT 3,1,47,15,35,84,74:PAUSE 1
        BLIT 3,88,47,15,35,84,74:PAUSE 1
        BLIT 3,175,47,15,35,84,74:PAUSE 1
        BLIT 3,1,124,15,35,84,74:PAUSE 1
        BLIT 3,88,124,15,35,84,74:PAUSE 1
        BLIT 3,175,124,15,35,84,74:PAUSE 1
      NEXT

* REM  picture of the boy blits on now...
      BLIT 4,2,60,5,4,264,136
      PAUSE 10
      MOVE 20,165
      TEXT "TAME THE ATOM FOR":MOVE 20,180
      TEXT "PEACEFUL PURPOSES"

* REM  fission animation begins here...
      FOR f=1 to 15
        IF f=5:BLIT 4,2,14,13,150,150,38 :ENDIF
        IF f=7:MOVE 20,165
*       PEN 1,15:CIRCLE 82,22,15
        TEXT "ORDINARY VINEGAR-":MOVE 20,180:ENDIF
        IF f=9:cycle 1:ENDIF
        IF f=11
          TEXT "FUEL FOR TOMORROW"
        ENDIF

        IF f=13:CYCLE 0:PALETTE 1,0:ENDIF
        BLIT 3,1,5,255,150,52,39:PAUSE 1
        BLIT 3,56,5,255,150,52,39:PAUSE 1
        BLIT 3,111,5,255,150,52,39:PAUSE 1
        BLIT 3,166,5,255,150,52,39:PAUSE 1
        BLIT 3,263,36,255,150,52,39:PAUSE 1
        BLIT 3,263,77,255,150,52,39:PAUSE 1
        BLIT 3,263,118,255,150,52,39:PAUSE 1
        BLIT 3,263,159,255,150,52,39:PAUSE 1
      NEXT
*     FADE 0,1,0
*     PAUSE 10
      END

```

Save it and run it:

```
director testscript
```

This should give you the feel of building a display and animation "film" with the Director. As long as you've gone this far with the example, you might continue to add your own changes and polish it even further. Images2 has a couple of blank screen parts left for you to paint into with your paint package, then work into the display. You might add opening titles to the sequence, and closing credits. Create your own animations in the boxes on "images1" and try them out with this script. All positions will already be set up to run smoothly. Add your own pictures and fonts, change the text. Create another screen full of images to add to the display. You might try a double buffering sequence following the method outlined in the manual. Read the section on sound, and add some sound effects. The more you experiment, the more familiar you will become with editing and altering scripts and displays. When you feel at home with the program, start from scratch and build your own film. Each time you create a new program, try a new command. Good luck, and let us see what you create.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15



ADVANCED TECHNIQUES



6. Advanced Techniques

Library Routines

On the Director disk, there are several library routines contained in the subdirectory "library". These routines can be included in your programs to do additional effects. The files themselves are well commented as to how they operate. Most of them you will use with the GOSUB command after perhaps setting up a couple of variables. A checkerboard wipe routine, a plane wipe routine, and a basic ANIM playback routine are included among other things. If over time, you yourself create special subroutines that you might like to use in scripts later, you might decide to split them out of where they were originally created to be included in a "library" for your later use. An attempt has been made to eliminate variable name and line number conflicts by choosing variables that all start with the same characters, and line numbers in the range 9000-9999.

Video Tape Recording Your Sequences

If you are primarily producing sequences that are to be videotaped, you will probably want to put the SETLACE command somewhere at the front of your script. The SETLACE command will turn interlace mode on, even if your images are low resolution. If you do not record the Amiga's output while in interlace mode, you may notice some vertical noise bars down the middle of your finished recording. The SETLACE command should cure this problem.

String Handling

You can manipulate character strings in various ways. The STRING command can be used to move strings into various places in the array. The COMPARE command can be used to compare text strings to see if they are equal, and provides a wildcard matching feature. And, by using the equals sign:

VALUE = \$(0)

You can convert a string that contains a numeric value into a variable that contains that value.

With COMPARE, you can test to see if a string contained in the array, is equal to something you're looking for:

```
COMPARE eq,"quit",$(20)
IF eq:PRINT "found a quit":ENDIF
```

Compare will return a 1 in the input variable "eq" if the string at \$(20) is exactly "quit". If you want to test for an uppercase "QUIT" you must specify that exactly. You can also look to see if it's just close to what you're looking for:

```
COMPARE eq,"qu*",$(20)
IF eq:PRINT "starts with qu":ENDIF
```

The asterisk tells COMPARE to not care what the rest of the string in \$(20) is, as long as it starts with a "qu", it compares as equal. You can even test to see if the character appears anywhere within the string you're testing:

```
COMPARE eq,"*qu*",$(20)
IF eq:PRINT "found a qu":ENDIF
```

In this case, as long as there is a qu somewhere in the string, you will get an equals result. The asterisk basically means "I don't care what is here".

The STRING command simply allows you to move strings around, or to setup strings in the array:

```
STRING "ABCDEFGHILMNOPQRSTUVWXYZ",$(50)
STRING $(20),$(50)
```

The first example will put the string of all the capital letters into the array starting at cell 50. The second example will copy the string that starts at cell 20 to cell 50.

You can then go in and modify the characters directly if you wish, by indexing them as numbers using the "@" sign:

```
@(50)=@(50)+32
@(51)=0
```

The numeric coding for the numbers that is used, is called ASCII. This stands for the American Standard Code for Information Interchange. The exact coding for these is in the back of your Amiga manuals. The capital letters are sequential and start with 65=A, and the lowercase letters are also sequential and start with 97=a. You can convert capital to small letters by adding 32, or small to capital letters by subtracting 32. See the ASCII chart in your Amiga manuals for details on punctuation marks, numbers etc.

The "tvgal" demo on the Director disk contains an example of the construction of random characters by using the string array. Basically, it is done like this:

```
ARRAY 100,1
FOR xpos=10 TO 200 STEP 10
  @(0)=65+?26
  @(1)=0
  MOVE xpos,20
  TEXT $(0)
NEXT
```

The first cell of the "@" array is stored a random number from 65 to 90. This represents the ASCII coding of the capital letters A-Z. To make it into a one character string, the zero that terminates a string must be put in the following cell in the "@" array. We then MOVE to a position on the screen, and use the TEXT command to output the single character string.

Strings tend to get used in conjunction with file input and output, and with keyboard input and output. Some of the examples in the next few sections will also contain examples of the use of the string handling commands.

File Input and Output

You can use an external text file for keeping more complex data used with a script. You can open files, read text lines into the @ array as strings, convert strings to numbers, and seek randomly throughout the

file. By building a test file with some arbitrary character string used as a record separator, you can scan an entire data file, locating the file positions of the beginning of the records and store them in the @ array for reference. Then you can use these references to randomly access the records. Tables of data can be input as well. There are six commands for performing file I/O. These are: OPEN, READ, WRITE, LOCATION, SEEK, and CLOSE.

OPEN

In order to use a file, you have to open it first. You do an OPEN something like this:

```
V=0
OPEN V,"filename"
IF V=0
    PRINT "File open error"
END
ENDIF
```

V is an example variable, and can be any variable name you wish. If V is zero, then you are doing an open for READING the file. If V is a 1, then you are doing an open for WRITEing the file. If V is a 2, then you are doing an open for both READ and WRITE.

After the open command, the variable V will contain a 1 if the open was successful, or a 0 if unsuccessful. The main reason for an unsuccessful open, is that the Director can't find the file from the name you have specified.

READ

Once you have opened a file for READ, you can then read it like this:

```
READ V,$(0),40
IF V=-1
    PRINT "End of file"
ENDIF
```

A read command like this, will read a line of text from the input file, and store the first 40 characters of the line, into the array at position 0. If an error occurs, V will be set to -1. The most common error in a READ will be that the end of file was reached. Note that at

least 41 cells would have to be reserved in the array for this command, as if there are 40 or more characters in the line read from the file, the 40 characters plus an ending 0 will be placed in the array.

Also note that the array must be defined by using the ARRAY command before such reading can occur.

WRITE

If you have opened a file for WRITE, it will create a new file with the name you have specified in the OPEN command, deleting any previously existing file by that name that may already exist. If you want to test if the file exists or not first, you can open the file for READ first, and if an error occurs, you can assume that the file does not already exist. Be sure to do a CLOSE on the old file before re-opening your new file.

The WRITE command takes a list of parameters similar to the PRINT command, separated by semi-colons. The WRITE command will construct a line of text based on the parameters of the command, and then write that line of text out to the file. A WRITE command sequence would look like this:

```
WRITE "Write this: ";N;$(20)
```

Any combination of parameters that is valid in the PRINT or TEXT commands is also valid in the WRITE command.

LOCATION

The LOCATION command allows you to determine where you are positioned within your opened file. It will return a number in a variable that you specify that is a character position of where you are presently "located" in your file. There may be times when you want to keep track of where various items of data start in your file, by saving the position in the file before they are written, to later be accessed by the SEEK command. The LOCATION command looks like this:

```
LOCATION V
```

SEEK

The SEEK command allows you to specify at what character position in your opened file you want to be positioned. This will allow you to re-read portions of the file that you may already have read or written. You have full random access to your text file, by SEEKing to any character position you wish. If you seek to position -1, you are specifying the end of the file.

CLOSE

Any time you OPEN a file, you must CLOSE it before you can open another file.

Using File Input and Output

One possible application for file input might be to keep a list of files to be used with an sequence in a separate text file so that they can be modified separately from the main script.

If we create a text file named "piclist" that looks like this:

```
:pictures/mountain
:pictures/tree
:pictures/house
end
5
```

A Director script that can read those in and load the pictures could work like this:

```
        ARRAY 100,1
        buff=1
        mode=0
        OPEN mode,"piclist"
        IF mode=0
            PRINT "Can't open piclist":END
        ENDIF
10      READ eof,$(0),80
        IF eof=-1:wait=10:GOTO 30:ENDIF
        COMPARE eq,"end",$(0)
        IF eq:GOTO 20:ENDIF
        PRINT "Loading file ";$(0)
        LOAD buff,$(0)
        buff=buff+1
        GOTO 10

REM got end, now let's read parameter
20      READ eof,$(0),80
        IF eof=-1:wait=10:GOTO 30:ENDIF
        wait=$(0)

REM now that we've got them all, let's page flip
30      FOR i=1 to buff-1
            DISPLAY i
            PAUSE wait
        NEXT
        GOTO 30
```

Input from the Keyboard

You can input text from the keyboard during your Director scripts. In order to do so, you can use the INPUT command. In order to use the INPUT command, you must make sure you have a displayed screen, and the array must be defined to provide a place to put the input text string.

Then you can try the INPUT command like this:

```
MOVE 10,10
INPUT $(0),40
```

The MOVE command specifies the place on the screen where to input from. The \$(0) specifies where within the @ array to place the string, and the 40 specifies a maximum of 40 characters. Since character strings are always stored with a zero at the end, up to 41 characters will be placed into the array starting at cell 0.

The INPUT command will input a line of text from the keyboard until a return key is depressed, and put it into the array for use by other Director commands.

Keyboard Input Via CLI

There may also be cases where it is desirable to input a line of keyboard information from the CLI. In particular, where a LOAD filename is not known, but needs to be asked of the operator, you can use a script something like this:

```
ARRAY 100,1
PRINT "What file to display?";
INCLI $(0),80
A=0
OPEN A,$(0)
CLOSE
IF A=0
    PRINT "There is no such file!"
    END
ENDIF
LOAD 1,$(0)
PAUSE 100
```

We first setup the array with the ARRAY command so we can use strings. The PRINT statment uses a trailing semi-colon so that a cursor-return is not performed before the INCLI. The INCLI will accept the users input until a return. We use OPEN just to see if the file exists. If the OPEN was unsuccessful, we can assume that there is no such file and report that to the operator. If we do get a good file name, we can then LOAD the file for display.

Other Input

Single Keystroke Input

Sometimes we may not want to input a complete line of text from the operator, but just a single key such as a function key or something, we can use the GETKEY routine. Getkey will get a single key from the operator and not display it on the screen.

Use GETKEY like this:

```
GETKEY c
MOVE 10,20
TEXT "The character was a ";c
```

After the GETKEY, the variable c will contain the ASCII numeric representation of the key that was hit.

Make sure that ABORT is not set to abort on a keystroke, as if multiple keystrokes are accidentally hit, the program may then see it as an ABORT command, and exit.

Other times, we may want to check and see if a key has been hit, but not wait forever for one. We can be doing a continuous loop of an animation, while periodically checking for a keystroke with the IFKEY command. Once a key is detected, we can then decide to go off and do a different animation, or other function. There is an IFKEY example in the following section on mouse input.

Mouse Input

Similar to GETKEY and IFKEY, there are the GETMOUSE and IFMOUSE commands. They will allow you to either wait for, or check for a mouse input to your animation screen. Again, be sure the ABORT command is set to disable mouse abort, so that the Director doesn't take your mouse click as a signal to exit.

In addition, you will want to turn the mouse pointer on with the command POINTER 1 if you are using IFMOUSE because the Director will start up

with the mouse pointer turned off. If you do not turn the pointer on, the operator will have no idea where he is pointing the mouse when you do your IFMOUSE. POINTER 1 is not required when using the GETMOUSE command. The Director is waiting specifically for a mouse command, and it is assumed that the pointer should be displayed. The Director will automatically turn the pointer off after a GETMOUSE has gotten its mouse data. POINTER 0 can be used to turn off a pointer display that was turned on with the POINTER 1 command.

Here is an example of IFMOUSE and IFKEY together:

```

        POINTER 1
10      IFMOUSE x,y
        IF x#-1:GOTO 20:ENDIF
        IFKEY c
        IF c#-1:GOTO 30:ENDIF
        PAUSE 2
        GOTO 10

20      ....do stuff with mouse x,y
        GOTO 10

30      ....do stuff with character c
        GOTO 10
```

Here we are checking if either a mouse click or keystroke has occurred. If a mouse click has occurred, we go to line 20, and can then process the click. If a keystroke has been made, we can go to line 30 and process the character. If neither has occurred, we keep looping and checking. A slight pause has been added, so that when nothing is happening, we relax a little so that other programs running along with our sequence using the Amiga's multitasking are not degraded by our continual processing. This is optional, but will allow better performance of other programs running at the same time as your Director script.

With this mouse input you can draw gadgets on your screen for the user to click on, detect if such a click has taken place, and act accordingly.

Executing AmigaDOS Programs

Sometimes it may be useful to run other AmigaDOS programs from within the Director. Using the EXECUTE command, you can use AmigaDOS COPY command to move files into RAM:, to run various sound programs or other things that you may want to happen in concert with your display sequences.

Here is an EXECUTE example:

```
EXECUTE v, "copy :pictures/demo#? ram:"  
LOAD 1, "ram:demo01.pic"  
.....
```

Keeping your image files in RAM: in some instances may be more efficient than trying to preload them all into buffers. If you don't need extremely fast access to them, if they are kept in RAM: disk they will take up less space than if they are loaded into a buffer in many instances. While doing a LOAD out of RAM: is not nearly as fast as doing a DISPLAY from another buffer, it is still quite a bit faster than loading from the floppy disk.

Drawing Commands

The Director provides a series of drawing commands that you can use to cause the Director to render images on the screen directly. This may allow you to dress up primarily text oriented displays with some graphics without having to load image files from disk, as well as other uses. The best way to get familiar with the drawing commands, is to start up a script with a load of a blank screen, that has been configured by your paint program to have an interesting palette. Then try the commands one by one, and see what they do. The MOVE command is used in conjunction with both the DRAW and TEXT commands to do the initial positioning for those commands.

The FILL command can be used to fill in solid areas of the screen. You can either fill an area that is enclosed by an outline in a specified color, or fill in all of an area that is presently a specified color.

Here are some drawing examples:

```
PEN 1,15
CIRCLE 160,100,40
FILL 1,160,100
PEN 1,10
FOR i=1 to 40
  MOVE 10+?300,10+?180
  DRAW 10+?300,10+?180
NEXT
```

There is a routine in the LIBRARY directory on the Director disk called "grid" that can be included in programs and used to draw grid lines.

Using BLIT to do Page Flipping

A powerful way to use the BLIT command is to setup a partial screen page flipping animation. This technique can maximize your memory usage where only portions of the screen actually contain animated images rather than the entire screen. In order to do this, decide what you want to animate, and using your paint program, divide a screen into rectangles that contain the different images that you want to "flip". Depending on the size of the image, you may be able to get 2, 4, 6, 8 or even more "frames" of your animation on a single screen. If necessary, you may be able to use multiple screens in this way to further multiply the number of frames in the movement.

Once you have designed your "flip" screens, decide on what you want to use as a "background" or "frame" screen, for whatever you want to have on the screen that does not include the animated portion. Here, you may decide to introduce color cycling animation, or other effects.

In some instances, if the animated images do not move across the screen but stay in one place, you may want to actually build the frame rectangles on your "flip" screens to include the background that is supposed to be there. This may make it easier to erase previous frames. While you can use transparent mode to BLIT an object onto a background, if the next "frame" of the object does not completely replace the previous frame, you may find that you have to erase portions of the previous frame by BLITting the background back over it. Though this can be done, and may be required for some animations, it pretty much means you will have to double buffer which can eat up at least one more CHIP screen buffer. If your animation sequence is stationary, you can build your "frame" rectangles to include the background, and NOT use

transparent mode, so that the new rectangles when being BLITed to the screen always completely replace the previous image. You may find you can get away without double buffering using this technique.

The supplied demo "tvgal.film" is an example of non-double buffered hi-res partial page flipping. With this technique, it is possible to do the entire sequence using only two hi-res screen files making it possible for it to run on a 512k machine.

In order to make positioning of your images easier for the BLIT command, you can use the supplied Director script "blitutil" to make it easier.

The BLIT Utility BLITUTIL

Supplied on your Director disk, is a little utility that is actually a Director script itself, called BLITUTIL. To run the utility, simply type:

```
projector blitutil.film
           or
director blitutil.film
```

The utility will ask you the name of a file to be used as your background image, and then ask you for the name of a "frame" objects file, as described in the previous section. Next, it will display the "frames" screen, and allow you to draw a box with the mouse around your first frame as stored on the screen. Note that background files and frame files do not have to be the same resolution, though mixing HAM files with non HAM files may not be very useful.

Blitutil will then allow you to use the mouse and cursor keys to position your rectangle on your background screen, as you want to see it during the animation. You can do several rectangles this way, and when you decide to quit, you will find that the CLI screen contains a list of the BLIT command parameters you will need to enter into the Director to accomplish the various selected moves.

Blitutil will guide you with specific instructions on its use. In addition, the Director script for Blitutil is provided so that you may see how some of the functions were performed. Blitutil is a good example of an interactive program created with the Director.

Changing BLITDEST and BLITMODE

The BLITDEST command allows you to redirect the output of virtually ALL blit, text, and screen drawing commands to a buffer other than the currently displayed one. This is particularly useful when double buffering, as you may want to setup the next frame invisibly, while the previous frame is the one being seen.

Using the BLITMODE command can produce a variety of unusual effects, though it is not a simple matter to determine exactly what effect a given mode produces. The BLITMODE command adjusts very low level blitter capabilities, modifying how it does its transfers. The discussion of BLITMODE in the command references section may give you some clue as to how it does its work, but you will need to know something about binary logical operations to be able to decipher it. A book on computer logic may be helpful if you are interested in pursuing the matter further. In addition, the Amiga hardware manual has a more explicit discussion of how the blitter works. Look for the discussion on "minterms" which is the Amiga term for the parameter being adjusted by the BLITMODE command.

Overscan Screens

For some presentations, it can be useful to use the Amiga's overscan mode to allow display of IFF images that extend all the way to the edges of the computer monitor or T.V. screen. With DPaint and other graphics programs, you can create screens which are larger than 320x200 in low resolution, or larger than 640x400 in high resolution. Many ANIM format files use a 384x240 size screen as a standard.

When overscan screens are used, your preferences setting for the positioning of the display on the video screen may need to be modified. By using the POSITION command, you can set the x and y position of the upper left corner of the screen to adjust for this. The default position parameters, -1,-1 (which actually give you a position relative to the current preferences setting) are setup for standard 384x240 ANIM files and 704x464 hi-res images. For other situations, you can adjust this to any position you like. A typical overscan position is something like 84,25.

Mixing overscan and non-overscan images is possible, but unless you reposition the screen centering as you change from overscan to

non-overscan and vice versa using the POSITION command, the results may not be favorable.

ANIM File Playback

The Director will give you frame to frame control over playback of ANIM files, so you can either just play an ANIM file straight, or add effects of your own on a frame by frame basis. We will talk about "frames" as individual images of a multi-image animation sequence.

You can also use the Director to chain multiple ANIM files, where you have external memory, but otherwise the biggest ANIM file you may be able to create and save is one that will fit on a floppy disk.

ANIM file frame data can be loaded into a "screen" buffer (even though it is not actually a "screen" buffer at that point, that is what we call them). Separate from that process, the first frame of an ANIM file is a standard IFF file, and can be loaded with the Director's standard LOAD or LOADFAST commands, as well as DPaint II and other programs.

In order to playback an ANIM file, you must first load the ANIM data using LOADANIM, and load the first frame using LOAD. Then you need to duplicate the first frame into a new buffer. ANIM files are designed to be double buffered.

To better understand ANIM mechanisms, let us define a little terminology. For this discussion, the CURRENT frame, is the one we are currently seeing on the screen, the PREVIOUS frame, was the frame we saw just before the current frame, and the NEW frame, is the frame we want to see just after the current frame.

We will then consider one of our two screen buffers, which should initially contain duplicate copies of our first frame, to be our "current" frame and our "previous" frame. At the start of the animation, the frame being displayed is the "current" frame, while the non-displayed frame will be considered the previous frame.

When the ANIM command is executed, the buffer number of the ANIM frame data is specified, and the buffer number of the "previous" frame also specified.

The ANIM command, will then modify the "previous" frame and turn it into the "new" frame. Then, after a reasonable delay due to a PAUSE command or other activity designed to provide the proper frame-to-frame timing

of the animation, we use DISPLAY making the "new" frame the "current" frame, and making the "current" frame the "previous" frame.

Let's try this with an example:

```
LOADANIM 3,"test.anim"
LOAD 1,"test.anim"
POSITION -1,-1
NEW 2,1
COPY 1,2
BUFF=2
10 ANIM 3,BUFF,ABS,REL,DONE
   DISPLAY BUFF
   BUFF=3-BUFF
   PAUSE 1
   IF DONE=0:GOTO 10:ENDIF
   PAUSE 100
   END
```

This short sequence will play an ANIM file once. If we want to continuously loop an ANIM file that has the correct 2 frame overlap (see the manual accompanying the software used to generate your anim files for more details), we can do it like this:

```
LOADANIM 3,"test.anim"
LOAD 1,"test.anim"
POSITION -1,-1
NEW 2,1
COPY 1,2
BUFF=2
10 ANIM 3,BUFF,ABS,REL,DONE
   DISPLAY BUFF
   IF DONE:SKIPANIM 3,BUFF,ABS,REL,DONE
   ENDIF
   BUFF=3-BUFF
   PAUSE 1
   GOTO 10
```

In this case, once the last frame is reached, SKIPANIM is used to skip over the first frame, as it is redundant. This ANIM playback will run forever, or until the mouse is used to ABORT the Director in the standard manner.

If we want to do other effects during the ANIM playback, we can subroutinize the frame-to-frame portion, and give us a little more flexibility:

```
LOADANIM 3,"test.anim"
LOAD 1,"test.anim"
POSITION -1,-1
NEW 2,1
COPY 1,2
BUFF=2
SPEED 1
50  GOSUB 10
    GOTO 50

10  PAUSE 2
20  ANIM 3,BUFF,ABS,REL,DONE
    IF DONE:SKIPANIM 3,BUFF,ABS,REL,DONE
    ENDIF
    BUFF=3-BUFF
    RETURN
```

With this approach, every time we do a GOSUB 10 or GOSUB 20, we will get the next frame of our ANIM animation. GOSUB 10 can be used where we aren't doing anything else, and want to pause the correct amount for the ANIM. Here we set the speed down so we can pause in smaller increments and make the sequence go faster if we like.

If we want, we can do a BLIT or other effects to either the "current" buffer or the "previous" buffer or both, in order to cause other things to happen during the ANIM playback. If other functions that we do take very long, we may find that those operations have taken enough time that the PAUSE is no longer necessary. In that event, we can use GOSUB 20 to generate the next frame without doing the PAUSE first.

If we want to do something very complicated at the same time, we may want to spread GOSUB 20 commands around through the other commands we are performing, so that we can keep the ANIM moving while our other commands take their required time.

Multiple ANIMs can be preloaded using LOADANIM and LOAD, and chained together end to end, or selected based on user input, a random number, or combined together in the same display using the BLIT command.

You do not have to start from the first frame of the ANIM file if you want. You can load the first frame into your paint package, modify it,

add color-cycling, etc., and save it out as a standard image file. For the playback, you can then load your image file as the first frame file, only using the ANIM file for the frame to frame animation, and gain a different color palette and/or color-cycling capabilities. Be sure not to save the loaded image out with the same name as your ANIM file, as the paint programs do not know to save the ANIM portion of the file, and will cause it to be lost.

The LIBRARY directory of the Director disk contains an ANIM file playback routine called "playanim" that can be inserted into a script and used to start off your scripts that need to display IFF ANIMs

Bit Plane Manipulation

The Director contains commands to allow individual manipulation of screen buffer bit-planes. The PLANES and ROTATE command can be used to copy and adjust bit-planes a plane at a time. With the PLANES command you can enable one plane only when doing text, and all the other planes when doing a graphic such as BLIT etc. This can allow you to keep your text and graphics on the screen simultaneously without interference. Or you can have your text plane cause all of your graphic planes to become darker just where the text resides, still managing the text and graphics independently.

In a low-res screen, there may be say, 5 planes. This allows you to have a 32 color palette. If you reserve the upper bit-plane for a text plane, then this means, that the upper 16 colors in the palette will only appear in the text, and which one will depend on what is on the other planes. If you make your entire upper palette (upper 16) the same color, then the text will always be the same color, no matter what the graphic is that's in the other planes. On the other hand, if you make the upper palette (upper 16) the same as the lower palette, but darker, everywhere the text appears, the graphic will seem to be darker.

The high order plane would be specified in the PLANES command with the number 16. PLANES 16 then, will enable only the "text" plane. The command PLANES 15 will enable all other planes and disable the text plane. If you do a ROTATE -1, this will move the text plane to be the low order plane. From here, you could do a BLIT from a single bit-plane buffer (which only has the low order plane) to the buffer, then do a ROTATE 1 to return this plane to the high order bit plane. This could allow you to do single plane page flipping onto a 5 plane image with a 4 plane graphic background.

Individual plane manipulation, while made easy with the Director, can present some confusing concepts. Unfortunately, these concepts imply some familiarity of binary arithmetic to be fully understood. A complete dissertation on binary arithmetic is beyond the scope of the Director manual. For more information, you can consult your bookstore or library. Look for a book on basic computer design, which should have a chapter on binary arithmetic. The Amiga hardware manual has a description of the concept of bit-planes as they apply to the Amiga.

The LIBRARY directory on the Director disk contains an example of single plane manipulation in the "planewipe" routine. This routine does a wipe, one plane at a time.



7

SOUND AND THE DIRECTOR

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

7. Sound and the Director

There are several ways you can add sound to your Director animations. With the sound module, you can add sound effects, and construct musical passages from sampled sounds. You can use the EXECUTE command to run other AmigaDOS commands that generate sounds. Or if you plan to videotape your animations, you can decide to add sound effects or music to the tape, after the animation is recorded.

The Sound Module

The sound module is a way to tie specific sampled sound events to your Director presentations. Sound in the Director is treated as a separate module because not all Director presentations will include sound. For presentations that do not use sound, not including the sound module will gain a bit of memory space that would otherwise be used by the sound module or its sound buffers.

Things To Think About

When considering how to add sound to your Director presentations, there are several things you need to think about. Memory usage will probably be an important issue. The Director can easily make use of all your available "chip" memory, some of which is also required for your sampled sound buffers.

If you have expansion memory, the memory taken up by the sound module itself may not be an issue, but individual sampled sound buffers are required to be in "chip" memory. If you want to do a presentation that uses a wide variety of sound effects and/or sound loops, you may find that this may impair some of the effects you wanted to do visually, because of the amount of chip memory you will have to set aside for the sounds.

If you decide to do a Director presentation that includes sound, it is good to decide this before you start your presentation, so you can keep in mind the fact that you may want to reserve some chip memory space for your sound samples.

For the most part, a sound sample will occupy as much memory in "chip"

ram as it occupies on disk, unless the sound sample is an IFF file in which case it may occupy less. When you load an IFF file, you specify which octave to load, allowing you to only load specific octaves if you don't need the entire set.

If you do not have expansion memory, then the sound module itself will take up about 20,000 bytes of your "chip" memory. With expansion memory, this 20,000 bytes is taken out of your expansion memory, and will not impact your chip ram.

What is "Sampled Sound"?

Sampled sound is a sound that has been "captured" by the computer, much like a tape recorder, but in a short segment that can be played at different speeds, and "looped" over and over as many times as you wish. There are several ways of obtaining sampled sound. Any of the audio digitizer peripherals for the Amiga can be used to construct IFF standard instrument sounds, which can be used. In addition, the Jukebox standard samples can be used. Jukebox samples tend to be long loops of several seconds that can be a phrase of music, a sentence spoken by someone, or a long chain of sound effects.

Sampled sound can give you the most realistic sound effects possible on the Amiga. Many of the music packages come with IFF instrument files that are compatible with the Director's sound module. The Director disk also contains some examples.

Using the Sound Module

To use the sound module, you need to include a MODULE command in your program somewhere before you need to use sound. The MODULE command will load the sound module, called "sound.mod" and included on the Director disk in the :options directory.

The MODULE command looks like this:

```
MODULE "sound"
```

Once you have started the sound module, you can then do SOUND commands that send specific commands to the sound module. The sound commands are LOAD, FREE, PLAY, QUIET, SLOWFADE, and KILL.

All sound commands are given to the sound module via the SOUND command, like this:

```
SOUND v, "LOAD", 1, 2, ":sounds/crash.snd"  
SOUND v, "PLAY", 1, 1, 63, 400
```

The first parameter in all SOUND commands is a variable, designed to return any result for that command. Not all commands return useful values in the specified variable, but it is required in all SOUND commands for consistency.

The LOAD sound command

The LOAD sound command allows you to preload a digitized sound file. IFF files and Jukebox format files are compatible. Unformatted sound files may be able to be loaded, you will just have to try them and see if the result is acceptable.

The syntax of the LOAD sound command is:

```
SOUND v, "LOAD", buff, octave, filename  
SOUND <variable>, <string>, <expr>, <expr>, <string>
```

"buff" specifies the sound buffer to use. Numbers 1-20 are acceptable. "octave" specifies which octave to load out of an IFF file which can be multi-octave. Octave numbers are not related to any musical octave, just the first, second, third, etc. octave as stored in the IFF file. Since IFF files store octaves starting with higher pitched octaves first, octave 1 of an IFF file will always be the highest pitch. Octave is an optional parameter. If omitted, the last octave (lowest pitch) will be used. The LOAD command will return a default "period" of the loaded sound. This is a value that can be used in the PLAY command to set the nominal speed of playback. This sort of speed setting will also have the effect of adjusting the pitch of the sound.

The PLAY sound command

The PLAY command allows you to play sounds you have LOADED. The syntax of the play command is:

```
SOUND v, "PLAY", buff, times, volume, period  
SOUND <variable>, <string>, <expr>, <expr>, <expr>, <expr>
```

The PLAY sound command will playback a sound buffer that you have loaded with the LOAD command. You specify the buffer number ("buff"), the number of times to play the buffer ("times") the volume to play at as a value from 0-63, and optionally, the period which adjusts speed and pitch. If "times" is zero, the buffer will play over and over until you do a QUIET or KILL command. "period" should probably be a value somewhere between 128 and about 2000. The smaller the number, the higher the pitch. If you omit the period command, the default period for the sound is used. You can play up to 4 sounds at a time, as there are 4 sound channels. Attempting to play a fifth sound, when four other sounds are presently playing, will do nothing.

Sound channels are allocated on an as-available basis. You can get some stereo effects, by doing two PLAY commands in quick succession, as they will usually be played out of different left/right channels. This is not guaranteed, but with a little experimentation, you will find that simple stereo effects can be performed.

The QUIET sound command

The QUIET sound command is simply:

```
SOUND v, "QUIET"  
SOUND <variable>, <string>
```

This forces all sound channels to stop immediately. This can be useful if you have set a sound or two to play indefinitely, and you then decide you want to stop and do something else.

The SLOWFADE sound command

This command allows you to tell the sound module to slowly fade out any sounds that may presently be being played. If you have set up any looping sounds, this command will automatically fade them out and then do the equivalent of a QUIET which will completely terminate the loops. The SLOWFADE command looks like this:

```
SOUND v, "SLOWFADE", speed  
SOUND <variable>, <string>, <expr>
```

The speed parameter will adjust the speed of fadeout, where 1 is the fastest, and 10-20 should be about the slowest you ever need.

During the time the sound module is doing a fadeout, it is not available for other commands. If you send it commands, they will not be processed by the module until the fadeout has completed. It is probably not a good idea to send the sound module a large series of commands during fadeout, as after fadeout, the command buildup may exceed the sound modules ability to process them effectively.

Since the sound module will be busy during sound fadeout, attempts to abort the Director or the Projector during a sound fadeout will result in a delay until the fadeout has completed. An abort by the Director or Projector will signal the sound module and wait for it to abort. The abort will not take place within the sound module until the fade has completed.

The FREE sound command

This command will free up the memory being used by a sound buffer. The syntax is:

```
SOUND v, "FREE", buffer  
SOUND <variable>, <string>, <expr>
```

This command should not be used to free up a sound that might be being played at the time, as distortion of the sound may occur.

The KILL sound command

This command will tell the sound module to go away. After using the kill command, you will not be able to issue any more SOUND commands, as there is no module there to process them. The memory being used by the SOUND module will then be freed. The freeing of this memory may take a second or two, so if you are killing the sound module in order to gain access to its memory, you may want to insert an appropriate PAUSE command after the kill command to allow the module time to clean up and exit.

The KILL command syntax is:

```
SOUND v, "KILL"  
SOUND <variable>, <string>
```

Upon aborting the Director, a KILL sound command is done automatically.

Running Other Sound Programs

By using the Director's EXECUTE command, you can run some of the other Amiga programs that generate sound of their own. For example, there is a public domain SMUS music player program, that can play back musical scores generated by several available music programs. By using the Director's EXECUTE command, in conjunction with the AmigaDOS RUN command and such a sound player, the sound track can be started up as a separate program from the Director.

The disadvantage to obtaining your soundtrack by running a sound player program with the Director's EXECUTE command is that you may not be able to synchronize the visual effects with the sounds very well. In many cases this may not be very important. In others, it may be very important. If an external sound player program does not take over all of the Amiga's four voice channels, it may be possible to combine such a player program with better synchronized effects from the Director's sound module. Exactly how they may work together you may have to discover by experiment. Not all sound programs behave themselves well when run together, and can compete with each other for the Amiga sound channels. The sound module is designed to be well behaved and not to take over any more sound channels that it needs, and then to do it in a way that is designed to be compatible with other Amiga sound programs. Not all programs may work that way, and it may take some experimentation to determine what limitations may exist when running the sound module in conjunction with other programs.



COMMAND REFERENCE



8. Command Reference

In following command reference section, commands are in alphabetical order, and generally conform to the following format:

```
commandname  parameternames
commandname  parametertypes
```

```
commandexample
commandexample
```

```
textdescription
```

The first line introduces the command, and names the parameters so they can be referenced within the text description. The second line shows the same command, but with the parameter types shown. Following that there are some examples of the use of the command, and then the text description. In parameter types, <expr> means any valid expression, <variable> means any variable, and <string> means a character string surrounded by quotes, or a string contained in the @ array and referenced by \$<expr>. <arrayindex> is an expression to be used as an index into the @ array. An <expr-list> is a list of a combination of string and numeric expressions separated by semicolons.

ABORT flag
ABORT <expr>

```
ABORT 0
ABORT 2
```

Will set abort flag. 0 means do not abort (you will have to reboot to quit the Director unless an error or an END occurs). 1 means a mouse click will signal quit to the Director (the default) 2 means a keystroke will quit the Director, and 3 means either a mouse click or keystroke will do a quit. ABORT should be used to disable mouse abort when IFMOUSE is used.

ANIM abuffer,pbuffer,abs,rel,end
ANIM <expr>, <expr>, <variable>, <variable>, <variable>

ANIM 3,1,abs,rel,end
ANIM 7,b,a,r,q

Generates the next frame of the ANIM data specified by "abuffer" by modifying the screen buffer "pbuffer". Will return absolute time (since the beginning of the ANIM) and relative time (since the last frame) if specified in the ANIM file, in variables "abs" and "rel" respectively. Variable "end" will be set to zero if the frame processed was not the last frame, and to one if the frame processed was last frame. ANIMs must be double buffered for best results, see chapter 6 for more information.

ARRAY cells,cellsize
ARRAY <expr>,<expr>

ARRAY 1000,1
ARRAY 200,4

Will allocate memory for the "@" array. "cells" signifies the number of elements in the array, and cellsize indicates the size of each cell in bytes. A size of 1,2 or 4 bytes are the valid size parameters. The "@" array cannot be used unless memory is allocated for it by the ARRAY command. If the "@" already exists, a second ARRAY command will cause the present memory reserved for the "@" array to be returned to the operating system, and the newly specified amount to be allocated, thus losing any information stored in the "@" array at the time.

The @ array is used thus:

@(<expr>) = <expr>

To assign an element of the @ array to a value.

Or, @ can be used in an expression thus:

$A+2*\textcircled{5}$

$\textcircled{(N+2)*5}$

Also, array elements can be used in the computation of array indices:

$\textcircled{N+\textcircled{5}}$

BLIT buffer,fromx,fromy,tox,toy,width,height

BLIT <expr>,<expr>,<expr>,<expr>,<expr>,<expr>

BLIT 2,0,0,xpos,ypos,32,27
BLIT buff,56,89,30,50,90,60

The BLIT command will transfer a rectangular area from picture buffer "buffer" at coordinates fromx,fromy to the "destination" buffer at coordinates tox,toy. Here, "coordinates" specifies the upper left corner of the rectangle. width and height specify the width and height of the rectangle transferred. The BLIT command allows you to use the Amiga's "blitter", which is special hardware designed for manipulating graphical data on the screen. While the BLIT command is not the only way from the Director to make use of the blitter, it is the most general mechanism for using the blitter to move portions of IFF pictures between buffers and to various locations on the screen.

The "destination" buffer is the current displayed picture buffer, unless explicitly directed to another buffer with the BLITDEST command. BLIT is quite fast, and can be used to do blit oriented animations. If transparency is selected with the TRANSPARENT command, then anywhere the source rectangle contains color 0 (the background color) no transfer occurs, allowing arbitrary shapes to be transferred.

With the use of the BLITMODE command, special blit operations can be specified that do more than a simple move. See the BLITMODE command for more information.

BLITMODE mode

BLITMODE <expr>

BLITMODE 51
BLITMODE -1

This sets the blitter mode to be used during BLIT, WIPE, and STENCIL. The blitter mode can be any value from 0 to 255. 192 is the default and means a straight transfer. All of the possible effects aren't documented here, (there are 256, though a somewhat less number are actually useful). One of the combinations is to effectively make your image into a negative, i.e. it will have the effect of inverting the color palette by inverting all of the pixel values causing all color 31 to become color 0, color 30 to be 1, etc.

BLITMODE is referred to as "minterms" in the Amiga hardware manual. BLITMODE -1 will set the blitmode to the default mode. Unfortunately, understanding of binary logical operations and binary arithmetic is required for full comprehension of the specific modes. The information on minterms here are provided for those who may understand binary logic operations and wish to experiment with specific capabilities of the Amiga hardware. If you do not understand binary logic, experimentation won't hurt, you can try all of the combinations from 0 to 255 searching for useful effects. Many of the modes may appear redundant or useless because they are only useful when performing specific blitter operations that reach beyond the scope of what is provided by the standard Director commands.

Minterms logical operations are:

minterms:	ABC	$AB\bar{C}$	$A\bar{B}C$	$A\bar{B}\bar{C}$	$\bar{A}BC$	$\bar{A}B\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}\bar{B}\bar{C}$
enable bits:	7	6	5	4	3	2	1	0

For the Director's BLIT commands, source "B" is the source bitmap, and source "C" is the destination bitmap. Source "A" is not used.

Here are some example blit modes:

$\overline{B}$ 00110011 (binary) 33 (hex) 51 (decimal)

Invoked with BLITMODE 51. This mode will set the destination bitmap to the inverse of the source bitmap. This is accomplished with the logical equation:

$$\overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C}$$

These are all of the logical terms selected by the binary number parameter. The first two terms reduce to:

$$\overline{A}\overline{B}$$

The next two terms reduce to:

$$\overline{A}B$$

And the combination reduces to:

$$\overline{B}.$$

$\overline{C}$ 01010101 (binary) 55 (hex) 85 (decimal)

Invoked with BLITMODE 85 This combination will set the destination buffer to the inverse of itself. The logical operations selected are:

$$AB\overline{C} + A\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}C$$

Which reduces to:

$$\overline{C}.$$

$\overline{B}\overline{C} + \overline{B}C$ 01100110 (binary) 66 (hex) 102 (decimal)

Selects logical operations:

$A\overline{B}\overline{C} + A\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC$

Which evaluates to the equivalent of EXCLUSIVE OR. Interesting effects can be obtained by continually exclusive or-ing a buffer to itself with a slight offset.

BLITDEST buffer
BLITDEST <expr>

BLITDEST 7
BLITDEST -1

Will set the destination screen buffer for subsequent blit operations to specified picture buffer. BLIT operations, MOVE, DRAW, CIRCLE CLEAR, etc. commands are all redirected to use specified screen buffer. This can be useful for offscreen manipulations and/or double buffering. The DISPLAY command will reset the blitdest to be the current displayed picture, but no other commands should affect its setting otherwise.

BUFFERS number
BUFFERS <expr>

BUFFERS 50
BUFFERS 100

BUFFERS will reconfigure the number of available screen buffers for a given Director program. The default is 30 buffers, but can be set to whatever memory will allow. The BUFFERS command must be used in a script before any buffers are actually LOADED with any of the load commands. In general, BUFFERS should probably be the first command in a script when it is to be used.

CENTER flag
CENTER <expr>

CENTER 1
CENTER 0

Turns auto centering on and off for the TEXT command. If flag=1, then centering is on, if flag=0 then centering is off. See the TEXT command for more details.

CIRCLE x,y,r
CIRCLE <expr>,<expr>,<expr>

CIRCLE 160,100,50
CIRCLE xval,yval,radius

x,y center, r radius Will do a circle using the foreground pen.

CLEAR
CLEAR

Clears screen to the background pen color.

CLOSE
CLOSE

Closes file opened by OPEN command.

COLOR buffer,permanence,colornumber,r,g,b
COLOR <expr>,<expr>,<expr>,<expr>,<expr>,<expr>

Will set palette color "colornumber" to color r,g,b where r,g,b are 0-15 color intensity commands for red-green-blue respectively. "buffer" is the buffer number affected, with -1 being the current screen. If "permanence" is a 0, the effect is temporary, will only affect the screen and not the copy of the palette stored with the screen buffer. If permanence is a 1, then the effect is permanent, and the buffer copy is updated.

COMPARE v,stringa,stringb
COMPARE <variable>,<string>,<string>

Will compare stringa with stringb, and set "v" to the result. Either stringa or stringb can be quoted strings, or string indexes into the @ array preceded by the \$ symbol. The "*" symbol in "stringa" specifies a match with anything in stringb.

For example:

COMPARE V,"*eat*",\$ (0)

Will set V to 1 if the letters "eat" appear anywhere in the string specified by \$(0), otherwise, sets V to 0.

COMPARE V,"Y*",\$ (0)

Will set V to a 1 if the string specified by \$(0) starts with a "Y". This can be useful for testing if the operator type in a yes or no, or just Y or N. Note that a comparison for "y*" should be used as well in case the yes was typed in lower case. "Y*" is not the same as "y*".

COMPARE V,"foo",\$ (0)

Will only set V to a 1 if the string specified by \$(0) is EXACTLY "foo".

COPY src,dest
COPY <expr>,<expr>

COPY 7,2
COPY -1,buff

Will copy src bitmap to dest bitmap without using the blitter if either of the bitmaps are not CHIP memory buffers. COPY will also copy the color palette. This is useful as a general buffer to buffer copy, and especially when the src or destination buffers are in fast memory, as the blitter will not operate with fast memory. The blitter will be used if both src and dest are CHIP buffers. If source or dest is -1, current displayed screen is used for that parameter.

CYCLE mode
CYCLE <expr>

CYCLE 1
CYCLE 0

If mode=1, turns on color cycling, if mode = 0, then turns it off. Note that if SETBLACK or FADE is used when color cycling is on, somewhat unpredictable results may occur, as these features will interact. If you are doing double buffering, or using wipes or dissolves, you may want to use the FIXPALETTE command to disable palette modification during related commands. If in doubt, you can try it either way to determine the effects as they apply to your situation.

DISPLAY buffer
DISPLAY <expr>

DISPLAY 2
DISPLAY buff

Select specified buffer as current displayed screen. Also sets blitdest to be selected screen buffer. Only CHIP memory buffers can be displayed with the DISPLAY command.

DISSOLVE buffer,fromx,fromy,tox,toy,width,height,speed
DISSOLVE <expr>,<expr>,<expr>,<expr>,<expr>,<expr>,<expr>,<expr>
<expr>

DISSOLVE 2,0,0,xpos,ypos,32,27,2000
DISSOLVE buff,56,89,30,50,90,60,20

Will do a dissolve from a rectangular area of buffer "buffer" to the current displayed screen. fromx,fromy is the upper left corner of the source rectangle, and tox,toy is the upper left corner of the destination rectangle. width and height are those of the rectangle. "speed" specifies the number of pixels to be transferred in a single blitter pass. A good range for a full screen dissolve is 1000-4000, with 3000 perhaps being an optimum choice. Larger numbers for speed cause the delay times between screen updates to increase so that they may be perceived as pauses. Smaller rectangle dissolves will usually use smaller speed values as they tend to run faster. After the dissolve is completed, the palette of the displayed buffer will be updated with the palette from the buffer dissolved from, unless the FIXPALETTE 1 command has been used to disable this feature.

DRAW x,y
DRAW <expr>,<expr>

Will draw a line using the foreground pen, from the current graphics cursor position, (specified by the MOVE command) to x,y. The graphics cursor position is then updated to be x,y. The line is drawn with the foreground pen (see PEN). To draw a line, try:

```
PEN 1,15
MOVE 10,10
DRAW 300,180
```

DRAWMODE mode
DRAWMODE <expr>

```
DRAWMODE 2
DRAWMODE -1
```

Sets the drawing mode to one of 3 modes:

- 0 - JAM1 (foreground only transfer)
- 1 - JAM2 (foreground and background transfer)
- 2 - COMPLEMENT (exclusive OR transfer, toggles screen pixels)

Mode 1 will put text on the screen in the foreground pen, with text background in the background pen color. Mode 0 will put text on the screen in the foreground pen, and put no text background on the screen at all. Mode -1 will set drawmode to the default mode (1).

ELLIPSE x,y,xrad,yrad
ELLIPSE <expr>,<expr>,<expr>,<expr>

ELLIPSE 160,100,80,40
ELLIPSE 200,50,50,30

Draws an ellipse with center at x,y, and x and y radii of xrad,yrad, using the foreground pen (see PEN).

ELSE
ELSE

If the previous IF instruction expression evaluated to zero, the instructions immediately following the ELSE instruction are executed. Otherwise, control is transferred to the instructions following the ENDIF instruction, after the instructions between the IF and the ELSE have been executed. The ELSE instruction is not required with the IF though an ENDIF is required for every IF instruction.

END
END

Causes the Director to end execution and clean up all screens and data spaces allocated.

ENDIF
ENDIF

Terminates an IF clause. See IF for more details.

EXECUTE variable,command
EXECUTE <variable>,<string>

```
EXECUTE a,"Assign fonts: mydisk:fonts"  
EXECUTE a,"run mydisk:c/smusplay mydisk:playfile"
```

This causes the Director to pass the "command" parameter to the CLI for execution as a CLI command. The result from the command execution is returned in the variable. Most commands do not return anything unless there is an error, though certain commands may return useful information that the Director can then work with. This command can be used to copy files into RAM:, delete files, run sound generation programs, anything that can be done by the CLI.

Though not required, it is probably a good idea when specifying filenames in EXECUTE commands, to use a complete path name, i.e. specify the diskname:directory/file you want completely, rather than use df1: etc. This is due to the fact that you may later be running your script under different circumstances, and not have the same disks in the same drives. By specifying the complete file names, regardless of which disks are in which drives, the program should still work correctly.

FADE flag,buffer,speed
FADE <expr>

FADE 1,7,0 (fade in quick using buffer 7 palette)
FADE 0,8,2 (fade out slow using buffer 8 palette)

Will fade in or out to/from the specified buffers color palette. If flag=1, will fade IN from a black state. If black is on (as from setblack) will result in a black off state after fade-in. Palette used is one from "buffer", and "speed" sets the duration of the fade. Useful speeds are in the range 0-10, with 0 being the fastest. If flag=0, will fade OUT (to black) with similar effects.

FILL mode,x,y
FILL <expr>,<expr>,<expr>

Will do a solid area fill of the area surrounding specified point x,y, with the foreground pen color. Valid modes are 0 and 1. Mode 1 does a flood of all surrounding points that match current point at x,y, and mode 0 does a flood fill of all surrounding points that don't match the outline pen (see PEN). As an example, to generate a filled circle, try this:

PEN 1,15
CIRCLE 160,100,60
FILL 0,160,100

FIXPALETTE flag
FIXPALETTE <expr>

FIXPALETTE 1 will disable any palette updating by the DISPLAY, WIPE, and DISSOLVE commands. This can be useful primarily when color cycling while double buffering, or doing wipes or dissolves. Otherwise, when these commands are done, they will update the palette which will have the effect of resetting the color cycle. FIXPALETTE 0 is the default, and enables automatic palette updating on these commands.

FOR variable=start TO end STEP increment
FOR <variable>=<expr> TO <expr> STEP <expr>

```
FOR inc=1 TO 7
FOR var=11 TO 88 STEP 11
FOR dcr=10 TO 1
```

where <variable> is a valid variable, and <expr> is a valid expression. Starts a loop using the variable specified. The variable is initialized with the value of the first expression. At the end of the loop, the value of the third expression is added to the variable, and the variable is compared with the value of the second expression. If they are equal, the looping is finished. If not, control returns to the statement immediately following the FOR statement. The end of a loop is marked with the NEXT statement. Every FOR command must be matched with a corresponding NEXT command. For example:

```
FOR K=0 TO 10 STEP 2
....
NEXT
```

will set K to 0, at the end of every loop will add 2 to K, and see if it has reached 10. If so, the looping is finished, otherwise it continues. The program will loop 6 times, with K being equal to 0,2,4,6,8,10 as it loops. The STEP value is an optional parameter, if omitted, a step of 1 or -1 (depending on direction to the TO parameter) is assumed.

If you exit out of a FOR loop without going through NEXT (by doing a GOTO) some information will still left on the looping stack. This can cause a stack overflow if done enough times during a single script.

FREE buffer
FREE <expr>

FREE 7
FREE buff

Deletes the specified screen buffer. This will free up all memory being used by the specified buffer so that the memory can be used for other things (other buffers, fonts, etc.). The current displayed buffer can not be freed. FREE will free either screen or ANIM buffers.

FREEFONT fontnum
FREEFONT <expr>

FREEFONT 2
FREEFONT font

Frees font loaded with LOADFONT specified by fontnum.

GETKEY variable
GETKEY <variable>

GETKEY char

Get a keystroke and assign it to the specified variable. i.e., GETKEY A will wait for a keystroke to be input, and assign the variable A to the resultant ASCII code of the key hit.

GETMAP buffer,arrayindex
GETMAP <expr>,<expr>

GETMAP 3,20
GETMAP 7,colortab

Will move a copy of the color map for buffer "buffer" into the @ array at the index specified by "arrayindex". Note that there is no "@" symbol in the array index specification, though the value is used as an index in the @ array. The map is a set of 32 2 byte numbers, each assembled from red,green,blue color codes in the following manner:

color = (red * 256) + (green * 16) + blue

Individual r,g,b color values can be extracted as follows:

red = color / 256

green = (color / 16) % 16

blue = color % 16

The map can be modified and replaced into a screen buffer with the SETMAP command.

GETMOUSE x,y
GETMOUSE <variable>,<variable>

GETMOUSE mousex,mousey

Will wait for a mouse click, and then insert the x and y location of the click into the two variables. The pointer will be turned on automatically if not already on from the POINTER command.

GETPEN variable,x,y
GETPEN <variable>,<expr>,<expr>

GETPEN color,32,20
GETPEN pixel,xval,yval

Will set "variable" to the pen value of the pixel at location x,y on the current screen, or BLITDEST specified buffer.

GOSUB linenumber
GOSUB <expr>

GOSUB 200
GOSUB 1000+N*100
GOSUB func

Go to subroutine command. Causes program to continue execution at line number specified. Subsequent RETURN statement will cause program to continue at line immediately following original GOSUB statement.

GOTO linenumber

GOTO <expr>

```
GOTO 300
GOTO 800+func*10
GOTO func
```

Causes program to continue execution at line number specified. Line numbers can be computed, or stored in variables.

IF condition

IF <expr>

```
IF 7=var-1:PRINT "equal":ELSE:PRINT "notequal":ENDIF
IF flag:GOSUB 100:ENDIF
IF char#-1:END:ENDIF
```

If condition evaluates to non-zero, the statements immediately following the IF are executed, otherwise, control is transferred to the next ELSE or ENDIF statement, whichever is encountered first. Examples are indented for clarity, indenting is not required.

Examples:

```
IF A=5
    MOVE X,Y
    TEXT "String"
ENDIF

IF (A<7)&(B>9)
    MOVE X,Y
    TEXT "String 1"
ELSE
    MOVE U,Z
    TEXT "String 2"
ENDIF
```

IFKEY variable
IFKEY <variable>

IFKEY char

Will set "variable" to either the ASCII value of a keystroke input from the keyboard, or -1 if none has been entered. Allows testing for keyboard input while doing other operations.

IFMOUSE x,y
IFMOUSE <variable>,<variable>

IFMOUSE xloc,yloc

Will set x,y to either the screen position of a mouse click from the mouse, or a -1 if none has been entered. Allows testing for mouse input while doing other operations.

INCLI \$(i),w
INCLI \$<expr>,<expr>

INCLI \$(0),30
INCLI \$(buff),8

Will accept input of a line of textual information from the keyboard. Will echo input keystrokes to the CLI window. Inputs the data into the @ array as a string at array index i. Array index must be preceded with the \$. Will only allow input of characters up to a maximum specified in "w". Handles backspace. <return> signals end of the line input.

INPUT \$(i),w
INPUT \$<expr>,<expr>

INPUT \$(0),70
INPUT \$(buff),12

Will accept input of a line of textual information from the keyboard. Will echo input keystrokes to displayed screen at the location of the graphics cursor (set with the MOVE command). Provides a text cursor of the underline symbol from the font. Inputs the data into the @ array as a string at array index i. Array index must be preceded with the \$. Will only allow input of characters up to a maximum specified in "w". Handles backspace. <return> signals end of the line input.

LOAD buffer,filename
LOAD <expr>,<string>

LOAD 1,":pictures/design1.pic"
LOAD buff,\$(0)

Will create buffer "buffer" and load IFF file "filename" into it. Loads only into CHIP ram, you must use LOADFAST to load into FAST ram. Any Director command that requires a current screen to operate, which includes all graphics, blitter, and text and input commands, can only function if a LOAD has occurred within the program that defines and creates the current screen. If you need a blank screen to work from, you can load a blank IFF screen with a palette setup for your work.

LOADANIM buffer,filename
LOADANIM <expr>,<string>

```
LOADANIM 2,":anim/test.anim"  
LOADANIM 21,$(animname)
```

Will load a standard picture buffer with all of the frame to frame information contained in an IFF ANIM file, for later processing by the ANIM and SKIPANIM commands. FREE can be used to free the memory space allocated by LOADANIM. ANIM data will automatically load into fast ram if such ram is available.

LOADFAST buffer,filename
LOADFAST <expr>,<string>

```
LOADFAST 2,":pictures/background.pic"  
LOADFAST buff,$(picname)
```

Will create buffer "buffer" and load IFF file "filename" into it. Loads only into FAST ram.

LOADFONT fontnumber,fontsize,fontname
LOADFONT <expr>,<expr>,<string>

```
LOADFONT 1,9,"topaz.font"  
LOADFONT 2,12,"ruby.font"
```

Loads font "fontname" size "fontsize" into memory and assigns font number "fontnumber" to it. Fontnumbers are used by the SETFONT command to specify which font to use for TEXT commands. Valid font numbers are 1-10. This command allows you to preload several fonts to be used, using SETFONT to select them from memory. Font memory can be released with the FREEFONT command. The fontname must contain the .font extension, i.e. "ruby.font".

LOCATION v
LOCATION <variable>

LOCATION loc

Assigns the current open file position to the variable "v".

MARGINS l,r
MARGINS <expr>,<expr>

MARGINS 20,300
MARGINS leftmarg,rightmarg

Sets text wrap points to "x" pixel coordinates l for left and r for right. Allows simple margin setting for text character-wrap. No word wrap is performed. Default margins are set at the time the initial screen is displayed, and margins are set at the edges of the screen.

When the centering is turned on with the CENTER command, text centering occurs between the two margins points set by the MARGINS command. Thus MARGINS can be used to adjust the centering to anywhere on the screen.

MEMORY all,chip,fast
MEMORY <variable>,<variable>,<variable>

MEMORY allmem,chipmem,fastmem

Sets the three variables specified to the amounts of memory in the system. The first variable is set to the total amount of memory in the system, the second to the amount of available CHIP ram, and the third to the amount of available FAST ram.

MODULE modulename
MODULE <string>

MODULE "sound"
MODULE "newmodule"

Specifies an expansion module to be used, such as the "sound" module. Once specified, the module can then be used. For information on individual module commands, see the section on the specified module. Modules are expected to be in an assigned "mod:" directory.

MOVE x,y
MOVE <expr>,<expr>

MOVE 32,70
MOVE xpos,ypos

Moves current graphics cursor position to x,y. This command is used in conjunction with the DRAW, INPUT, and TEXT commands.

NEXT NEXT

Terminates a FOR loop. FOR loops can be nested. A variable specification is not required as in some BASICs, though it can be used for clarity.

NEW buffer1,buffer2 NEW <expr>,<expr>

```
NEW 5,1  
NEW 7,-1
```

Creates a new CHIP memory buffer, "buffer1" using a copy of the palette for buffer "buffer2".

NEWFAST buffer1,buffer2 NEWFAST <expr>,<expr>

```
NEWFAST 10,2
```

Creates a new FAST memory buffer, "buffer1" using a copy of the palette for buffer "buffer2".

OPEN openmode,filename
OPEN <variable>,<string>

```
mode = 0:OPEN mode,"mydisk:testfile"  
mode = 1:OPEN mode,"ram:tempfile"
```

Opens file "filename" based on the value in the variable "openmode". The variable can have values as follows:

```
0 - Open for read  
1 - Open for write  
2 - Open for read/write
```

After the open, the variable specified as "openmode" will be set to a 1 if the open was successful, and a 0 if not. Only one file can be opened at a time. A previous file must be CLOSED before a new file can be OPENed.

If a file is opened for read/write, you will be positioned to the end of the file. This will allow you to append data to an existing file. By SEEKing to other locations in a file, you can write over data that was there before with new data.

PALETTE buffer,permanence
PALETTE <expr>,<expr>

```
PALETTE 2,1  
PALETTE 7,0
```

Sets the screen palette or BLITDEST buffer palette to the palette from buffer "buffer". If screen buffer is specified and "permanence" is 1, then both the current screen colors and current screen buffer's palette are modified, if "permanence" is 0, then only the current screen colors are modified. Buffer of -1 specifies the current screen buffer.

PAUSE v
PAUSE <expr>

PAUSE 20
PAUSE delay

Pauses for a time period specified by "v". The default time increment is .1 sec. (set by the SPEED command). This means that the pause will take v*.1 seconds, i.e. PAUSE 10 will pause for 1 second. The SPEED command can adjust the increment.

PEN pennumber,colornumber
PEN <expr>,<expr>

PEN 1,8
PEN 0,color

Sets selected pen to the specified color register. This will affect text, fill, and draw commands. Pens are as follows:

- 0 - Background Pen**
- 1 - Foreground Pen**
- 2 - Outline Pen**

There are 32 possible color registers, 0-31. Not all IFF image files may understand all 32 at a time. Different resolution screens may permit a fewer number of colors. The number of colors in a paint program palette for a given picture usually translates to the number allowed with the PEN command when displaying that picture from the Director. If using halfbright mode values 0-63 are valid, where colors 32-63 will be half the brightness of colors 0-31.

PLANES flag
PLANES <expr>

PLANES 4
PLANES -1

The PLANES command allows you to specifically enable which bit-planes are to be affected by other commands. With the PLANES command you can enable only specific planes for modification, preserving the data present on the remaining planes. The PLANES command will affect the DISSOLVE, WIPE, BLIT, TEXT, CLEAR, and drawing commands. Use PLANES -1 to enable all the planes.

Enable individual planes like this:

for plane 0: PLANES 1	for plane 3: PLANES 8
for plane 1: PLANES 2	for plane 4: PLANES 16
for plane 2: PLANES 4	for plane 5: PLANES 32

To enable planes in combinations, add the PLANES parameters together. For example, to enable plane 2 and plane 4, do a PLANES 20 (value for plane 2 is 4, and for plane 4 is 16, 4+16=20).

POINT x,y
POINT <expr>,<expr>

POINT 38,40
POINT xloc,yloc

Will plot a single point at position x,y. Uses the foreground pen for the color of the point.

POINTER flag
POINTER <expr>

POINTER 1
POINTER 0

Turns the mouse pointer on and off. The default for the mouse pointer is off. If flag = 1, then the pointer will be turned on, if flag=0 then the pointer will be turned off. Particularly useful when using the IFMOUSE command, as unless the pointer is turned on, it will be invisible.

POSITION x,y
POSITION <expr>,<expr>

POSITION -1,-1
POSITION 98,25

Allows setting of the position of the upper left corner of the display screen for overscan screens. If you are going to use standard 352x240 overscan screens, you can set the position to -1,-1 which is an estimated centering for the image you are using. For other size screens, or if you desire other adjustments, you can adjust the centering of the screen accordingly. 98,25 will probably be the most upper left position you should ever have to set. Adjusting this parameter is the same as manipulating the "screen centering" control in Preferences, except when the Director program exits, the centering is set back to where it was originally set when the Director started.

PRINT
PRINT <expr-list>

Displays data to the current CLI window. If <expr-list> is omitted, then a newline is printed. If the <expr-list> is included, the values of the expressions are printed. An <expr-list> can be a combination of <string> and <expr>'s separated by semi-colons ";". If a semi-colon is left on the end of the <expr-list> then no trailing return will be output.

Examples of valid PRINT commands:

```
PRINT "Hello"

PRINT "The value of X is:";X

PRINT "The string is " ;$(0)

PRINT

PRINT T;" , ";

PRINT "What is the filename?";
```

```
READ v,$s,w
READ <variable>,$<expr>,<expr>
```

```
    READ pos,$(data),80
    READ pos,$(0),30
```

Reads the data file if open, into the @ array at index "s", not to exceed "w" characters. If successful, "v" will contain the file position, if not successful, "v" will contain a -1. -1 usually signals end of file.

```
RECT ux,uy,lx,ly
RECT <expr>,<expr>,<expr>,<expr>
```

```
    RECT 20,30,60,90
```

Draws a solid color rectangle in the foreground pen color of the area bounded by upper left point ux,uy and lower right point lx,ly. If the outline pen is non-zero, will outline the rectangle with the color of the outline pen.

```
REM
REM
```

```
    REM ignore this message
    REM this is how this program works
```

The REM command (short for remark) allows you to place comments in your scripts to help remind you what is going on. Any text following the REM command on the command line is ignored.

RESOLUTION buff,width,height,planes
RESOLUTION <expr>,<variable>,<variable>,<variable>

RESOLUTION 2,wid,hite,plns
RESOLUTION -1,xsize,ysize,depth

The **RESOLUTION** command allows you to determine the resolution of any screen buffer. The "buff" parameter specifies which buffer, and the three variables following are set to the screen width, the screen height and the number of bit-planes respectively. -1 can be used as a buffer specification to refer to the current displayed screen.

RETURN
RETURN

Returns control to the statement immediately following the most recent **GOSUB** statement. Subroutines can be nested, that is, a subroutine can call a subroutine that calls a subroutine, etc.

ROTATE buff,amount
ROTATE <expr>,<expr>

ROTATE -1,1
ROTATE 2,-3

The **ROTATE** command will rotate the positions of the individual bit-planes within the specified screen buffer, 'buff'. This can have unusual effects on the palette (though it doesn't actually change, the individual pixels are what get changed). A more practical use for this command is in conjunction with the **PLANES** command to allow copying of image information from any plane in one image to any plane in another image. If the "amount" parameter is positive, it causes the low order bit-plane to move to the high order bit plane (right binary shift). If the "amount" parameter is negative, it causes the high order bit-plane to move to the low order bit plane (left binary shift).

SEEK n
SEEK <expr>

SEEK 0
SEEK recloc

Moves current file position to position "n". File must be opened with the OPEN command. -1 will specify the end of the file.

SETBLACK flag
SETBLACK <expr>

SETBLACK 1
SETBLACK 0

Will set the screen to black, or turn colors back on to the appropriate palette. Works by setting all the colors in the present screen palette to black. Will not affect actual palette associated with storage of screen. SETBLACK 1 turns screen off, SETBLACK 0 turns screen on. If set before initial LOAD command, first screen will come up black. A fade in can then be done, or a simple SETBLACK 0 to turn on the screen when desired.

SETFONT n
SETFONT <expr>

SETFONT 2
SETFONT 7

Sets the current font to preloaded font number "n". Allows the selection of several preloaded fonts. Fonts are designed to be preloaded to allow the avoidance of disk operation which might slow the change in font selection at critical times otherwise.

SETLACE
SETLACE

SETLACE will turn interlace mode on. SETLACE is most useful when some of the images used by a given Director script are to be interlaced. When an interlaced file is loaded, interlace is automatically turned on, but it may be desirable to turn SETLACE on early so the entire script will be run with interlace on. Interlace should be turned on in scripts where the output is to be videotaped, for best results. Interlace cannot be turned off once it is turned on.

SETMAP buffer,index
SETMAP <expr>,<arrayindex>

SETMAP 7,index
SETMAP -1,0

Sets the color map for screen buffer "buffer" to a color map in the @ array at index "index". Note that the "@" symbol is not used in the array index specification, though the value is used as an index into the @ array. If "buffer" is the current screen, the current screen colors will be affected. "buffer" of -1 can be used to specify the current screen. SETBLACK 1 will disable SETMAP's effect on the current screen colors, though a subsequent SETBLACK 0 will then display the newly selected colors.

SKIPANIM abuffer,pbuffer,abs,rel,end
SKIPANIM <expr>,<expr>,<variable>,<variable>,<variable>

SKIPANIM 2,7,abs,rel,done

SKIPANIM parameters are identical to the ANIM command parameters, except no modification to screen buffer specified by "pbuffer" is actually performed. SKIPANIM allows you to skip over frames that may be redundant, as in the standard 2 frame overlap ANIMs. See the ANIM command and section 5 for more details.

SOUND var,command,param,param,param
SOUND <variable>,<string>,<>,<>...

```
SOUND period,"load",buff,octave,":sounds/splash.samples"  
SOUND a,"play",buff,times,volume,period  
SOUND a,"quiet"  
SOUND a,"slowfade",speed  
SOUND a,"free",buff  
SOUND a,"kill"
```

The SOUND command is a module command. All module commands start with a variable which returns the result of the command, followed by a text string which is the command to be sent to the module. Parameters following these are dependent on the module command being sent. For more information, see the section on the sound module.

SPEED s
SPEED <expr>

```
SPEED 1  
SPEED -1
```

Set global speed to "s". The default is 5, which causes pause counts to be taken as .1 sec increments. Set speed to 50, and pause will be in 1.0 seconds etc. Set speed to 1, and pause will be in 1/50 second increments. You can change this as often as you want during a script. A speed of 0 causes all PAUSE commands to do nothing. SPEED can be used to affect ALL pause commands identically, thereby modifying the speed of the entire sequence or over portions of the sequence.

STENCIL picturebuffer,maskbuffer,maskpolarity
STENCIL <expr>,<expr>,<expr>

STENCIL 2,2,1
STENCIL 3,7,0

Does a masked "blit" of the entire buffer area. Buffer "picturebuffer" will be transferred to the current screen (or current BLITDEST buffer) at all points where "maskbuffer" is color 0 if maskpolarity is 1, or at all points where "maskbuffer" is NOT color 0 if maskpolarity is 0. "picturebuffer" and "maskbuffer" can be the same buffer if desired. **STENCIL** can be used for special effect "wipes" by generating patterns into a maskbuffer and then using **STENCIL** to transfer an image to the screen masked by this buffer.

STRING s1,s2
STRING <string>,\$<expr>

STRING "message",\$(40)
STRING "text data",\$(str)

Moves string s1 to s2. s1 can be either a quoted string or a string index into the @ array, but s2 can ONLY be an index into the @ array.

STYLE style,spacing
STYLE <expr>,<expr>

STYLE 2,0
STYLE 6,2

The **STYLE** command sets text font styling and inter-character spacing. Styles are:

0 - Normal
1 - Underline
2 - Bold
4 - Italic

Styles can be added together to produce compound styles, i.e. bold and italic is style 6. Inter character spacing is normally 0, but can be set to the number of extra pixel lines to be inserted between each character.

TEXT
TEXT <expr-list>

MOVE 10,10:TEXT "Figure B"
MOVE 40,80:TEXT "The answer is: ";X

Displays text data to the current screen buffer or BLITDEST buffer. The values of the expressions in <expr-list> are printed. An <expr-list> can be a combination of <string> and <expr>'s separated by semi-colons ";". If a semi-colon is left on the end of the <expr-list> then no trailing newline will be output. **TEXT** will use the currently selected font. The text color will be the foreground pen, and the background of the text will be the background pen. By using **DRAWMODE** you can set the text mode that can cause the background to be transparent. If centering is turned on with the **CENTER** command, text will automatically be centered between the margins set with the **MARGINS** command or default of the edges of the screen if no **MARGINS** command has been used.

Examples of valid TEXT commands:

```
TEXT "Hello"
```

```
TEXT "The value of N is: ";N
```

```
TEXT N*2
```

```
TEXT N*2;U*3
```

TRANSPARENT flag **TRANSPARENT <expr>**

Sets transparency mode on and off for the BLIT command. If you do a TRANSPARENT 1, BLIT commands will only transfer that portion of the specified rectangle that contains colors that are non-color-zero. TRANSPARENT 0 sets it back to normal. When transparency is turned on, the Director must allocate a "mask" bit plane, a 1 bit plane of the same resolution as the source buffer. If you do not have enough memory for this mask plane at any given time, the use of transparent mode can possibly cause out of memory errors during a TRANSPARENT, DISPLAY, or BLIT commands. TRANSPARENT 0 does not actually deallocate the mask plane, TRANSPARENT -1 can be used to force deallocation of the mask plane memory in the event that memory space is tight and you want to make sure the mask plane memory has been freed up for other use.

TROFF **TROFF**

Turns trace mode off.

TRON **TRON**

Turns trace mode on. Trace mode will display each command to the CLI window as it is executed, with the parameters as evaluated. The commands are slightly abbreviated, as that is how they appear in the internal command buffer, but should be enough to identify them. Trace mode only works with the Director and not the Projector. Trace mode can also be turned on for the entire script when the Director is invoked by typing "Director -t scriptfile" or "Director scriptfile opt t".

WIPE buff,sx,sy,dx,dy,w,h,amt,mode
WIPE <expr>,<expr>,<expr>,<expr>,<expr>,<expr>,<expr>,<expr>

WIPE 2,0,0,0,0,-1,-1,80,0
WIPE 4,20,20,40,40,60,60,60,3

Will slowscan some or all of a buffer to a specified position on the screen buffer.

Parameters are:

buff	screen buffer to scan FROM
sx,sy	source buffer upper left x,y of rectangle to be scanned.
dx,dy	destination buffer upper left x and y of rectangle to be scanned.
w,h	width and height of rectangle to be scanned. -1 will select the current screen width or height.
amt	number of pixels to scan in a single pass. This will directly affect the speed of the resultant scan. If amt=w, then an entire pixel line will be transferred in a single pass, resulting in a fairly fast wipe. If amt=1, will result in a fairly slow wipe.
mode	Specifies direction of wipe. Valid values are: 0 - Top to bottom wipe 1 - Left to right wipe 2 - bottom to top wipe 3 - right to left wipe

At the end of the WIPE process, the palette for the display buffer is updated with the palette from the buffer wiped from, unless the FIXPALETTE 1 command has been used to disable this feature.

WRITE
WRITE <expr-list>

WRITE answera;", ";answerb
WRITE "\$";value

WRITE will write a line of text to a text file if it has been opened with the OPEN command with the open flag set to write, or read-write. The format of the <expr-list> in the WRITE command is identical to that of the PRINT command.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

APPENDIXES

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Appendix A The ED Editor

ED is a simple text editor that comes with your Amiga. It is found in the c: directory on your workbench disk. To use ED to edit a script file, make sure you CD to the directory where you want your script file to be, and from the CLI type:

```
ed filename
```

Where "filename" is the name of the script file you are working with. The first time you use ED to do this, it will create a file with the name you specify. When you do this later, it will realize that the file already exists, and call it up for editing.

You can then type your script onto the screen. To save the file, type the escape key, and then "x" followed by a return. You can use the cursor keys to move around in your script. The backspace and delete keys are also useful while editing in ED. There are several other commands you can type in ED to help you in editing. Some of them work as control key commands, which means you hold the control key down like a shift key, and then type the character associated with the command. The main control commands are:

```
ctrl-a  open a new line after this one.  
ctrl-b  delete current line  
ctrl-g  repeat last ESC command  
ctrl-y  delete to end of line
```

There are also several commands that you prefix with the escape key, similar to the "x" described earlier. For all of these commands, first type the ESC key, then type the command as specified, followed by the return key. The ESC commands are:

ESC x	save file and quit
ESC q	quit without saving
ESC f/string/	"find" string
ESC bf/string/	backwards find string
ESC bs	mark block start
ESC be	mark block end
ESC wb/file/	write block to "file"
ESC db	delete block
ESC ib	insert block
ESC j	join line with next
ESC m#	move to line "#"
ESC b	go to bottom of file
ESC t	go to top of file
ESC if/file/	insert file "file"

Appendix B Using the CLI

Here is a list of the most useful CLI commands. For more information, you can find a book on AmigaDOS at your local bookstore or computer store.

CD

The CD command displays the current selected directory if you type it by itself, if you follow the CD with a specific directory name, it will select that directory as the current one.

COPY

The COPY command will copy files. "copy df0:tree.pic df1:" will copy the file "df0:tree.pic" to the home directory on drive 1.

DATE

DATE when run without parameters, will display the date. If you type the date and time following the DATE on the command line, it will set the date and time to that specified.

DELETE

DELETE is used to delete files and/or directories. "delete tree.pic" will delete the file "tree.pic" if you specify a directory name, and follow it with the word ALL, then DELETE will delete all of the files in the directory and then the directory itself.

DIR

DIR will list the contents of the current directory on the screen. You can use the spacebar to stop the output and the return or backspace to restart it. If you type "dir df1:pictures", DIR will list the contents of the directory "df1:pictures".

DISKCOPY

DISKCOPY will copy one disk to another "diskcopy df0: to df1:" will do an entire disk copy from drive 0 to drive 1.

ED

ED is a simple text editor that comes with your Amiga that can be a useful tool in editing your Director scripts. See Appendix A for more details.

EXECUTE

Execute causes a text file of AmigaDOS commands, such as the startup-sequence file in the s: directory, to run. Type "execute file" to run the AmigaDOS commands that are in the file "file".

FORMAT

FORMAT will prepare a completely blank disk that you can then copy files to. Type: format drive df1: name "mydisk" to format a blank disk in drive df1: and name it "mydisk".

INFO

INFO displays information about how full the floppy disks and RAM: disk are.

LIST

This command is similar to DIR, except it will give you more detailed information.

MAKEDIR

This command will create a new directory on a disk. Type `makedir dirname` to make a directory named "dirname".

RELABEL

This command changes the name of a disk, type: `relabel df1: "newname"` to rename a disk "newname".

SAY

Reads a specified text file and outputs it using the speech synthesizer program. You can include this command in a Director script by using the EXECUTE command in the Director. Type: `say filename` to have it speak the text file "filename". Note that you must have the `narrator.device` in the `devs:` directory and the `translator.library` in the `libs:` directory on your boot disk for this to work.

TYPE

TYPE will type out a file to the screen. Type: `type filename` to output the file "filename" to the screen. Use the space bar to stop and return or backspace to restart if it goes by too fast.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Appendix C Error Messages

If Director encounters an error during the process of running your script, it will display what was wrong on the CLI screen and exit in most cases. One common error will probably be out of memory while trying to do something. If this occurs, you will have to re-arrange your use of memory in some way, either try using the LOADFAST and COPY command or not preload so many images.

The trace mode is an invaluable aid in determining what is going on if your script is not working.

Also note that you can get more "frames" into memory if the individual frames have fewer bit planes. The first frame loaded however will specify how many bit planes to use for the screen display, so be careful when attempting to mix the number of bit planes used in a sequence. Mixing screens with different bit planes is possible, but may not always be useful.

Check out the sample script files that should provide examples of how the commands work and can be used to do various effects.

Error Message Detail

Many error messages will be output with the command name at the beginning of the message. To find the detail description of the error in this list, take the message following the command name, and look it up alphabetically.

Aborted

This is not an error. Your script has aborted because either an END command has been reached, or abort has been signaled by the operator either by mouse click or keyboard abort (as selected by the ABORT command).

Argument Must Be String Array

An argument to one of your script commands is expected to be a string array, and was found not to be. The INPUT and INCLI commands for example, require a string array specified as \$(index) where index can be

any expression.

Array Index Out of Range

An array index, specified as @(index) where index can be any expression, results in an index that is outside the actual size of the array as define with the ARRAY command, or the array has never been defined.

Buffer Size Mismatch

This error usually occurs during the COPY command, when trying to copy one picture buffer to another where the X and Y resolution of the two picture buffers differ. COPY can only copy between same size buffers, though differing number of bitplanes is allowed. The BLIT command can be used to copy partial pictures from one CHIP buffer to another, which may provide an alternative. Using DPaint or some other tool to adjust the resolutions to be the same may be helpful as well.

Can't Open Diskfont Library

The diskfont.library file is not on the system boot disk.

Can't Open _____.font

The system can't find the font specified. This may be due to having booted from a disk which does not contain the font. If the font is on another disk, the AmigaDOS command "Assign" can be used to tell the system where the correct fonts are. If you are packaging a disk for distribution, you may want to consider including a Director EXECUTE command to cause the Assign to be performed automatically. More information on how to do this can be found in Chapter 6.

Divide By 0

An attempt to divide by 0 has been made in an expression.

Dest Must be Array

The destination parameter of the `STRING` command must be an array specification, and not a quoted string.

File Not Open

An attempt to do a file operation where a file has never been opened has occurred. The `READ`, `WRITE`, `SEEK`, and `LOCATION` commands can only be performed if an `OPEN` command has been successful. If you are doing an `OPEN` command, make sure that the open was successful by checking the variable parameter afterwards for the correct indication. See the `OPEN` command in the command reference section for details.

Invalid Cell Size

You have an invalid cell size parameter to the `ARRAY` command in your script. In the `ARRAY` command, you can only specify cell sizes of 1, 2, or 4.

Math Stack Overflow

An overly complex expression that produces enough intermediate values has overflowed an internal arithmetic stack. This can be cured by breaking the expression down into two or more separate expressions.

Math Stack Underflow

This could occur as a result of a parenthesis imbalance.

Missing Parameter

An expected parameter was missing from a command in your script.

Next Without For

An attempt has been made to do a NEXT, where a FOR has never been done. FOR and NEXT must be matched pairs, check your program to make sure this is the case.

Nonexistent buffer

A buffer number parameter to one of the commands specifies a buffer that either has never been LOADED, (or was loaded and later FREEd) or a buffer number that is outside the range of allowable buffers. Using the BUFFERS command to adjust the range of allowable buffers, or making sure the screen is LOADED, or that the number is being specified correctly are the most likely options.

Nonexistent Line

An attempt to GOSUB or GOTO to a line that does not exist has been made. Recheck your script for correct line numbering, or correct expression associated with the GOSUB.

Not An ANIM Buffer

An attempt has been made to do an ANIM command where the anim buffer specified in the command is a screen buffer.

Not a Version 1.2 Film File

The .film file you are trying to run is not compatible with the version of the Director you are using, or is not even a .film formatted file. You may have this problem if you are running old prerelease demos or film files that were created with earlier (or later) versions of the Director. Either recreate the .film file from the original script with your version, or try and find the version of the Projector that the .film file was distributed with. Another way you may get this error if you have tried to name your script file with a name that ends in .film. The Director program will create your .film files for you, you should not attempt to name other files ending in .film

No Screen

A command that requires a screen to have been LOADED has been run before a LOAD has been done. The TEXT, INPUT, and GETMOUSE commands for example, require a screen to operate correctly.

Out Of Memory

You do not have enough CHIP or FAST (or both) memory on your system to do an operation. LOAD, LOADFAST, NEW, NEWFAST, DISSOLVE, TRANSPARENT, and STENCIL are all commands that are likely to run out of memory if you are manipulating a lot of things in memory with your script. The trace mode can help you determine exactly which command is causing this problem if it isn't obvious otherwise.

Parameter Not a String

A parameter that was supposed to be a string was found not to be, or was missing from a command in your script.

Parameter Not a Variable

A parameter that was supposed to be a variable was found not to be, or was missing from a command in your script. Some commands place results in variables, and in these cases, the parameter MUST be a variable so that this can occur. Numbers or expressions are not valid in these cases. See the exact command syntax in the command reference section for the suspect command.

String Index Out of Range

A string index, specified as \$(index) where index can be any expression, results in an index that is outside the actual size of the array as defined in the ARRAY command, or the array has never been defined.

Stack Overflow

FOR loops and/or GOSUB/RETURNs are nested too deep, or the program has exited a FOR loop without going through normal NEXT completion too many times. If you exit a FOR loop with a GOTO and never come back, loop stack information associated with the FOR remains on the stack, and if done enough, can cause a stack overflow error.

Return Without Gosub

An attempt has been made to do a RETURN, where a GOSUB has never been done. Make sure the end of your program doesn't keep going after the end of the main program and fall right into your GOSUBs. And END statement at the end of the main program can effectively separate the end of your main program from any following GOSUBs.

IF: Unmatched

An IF statement has no matching ENDIF statement. The ENDIF must occur sequentially following the IF, and cannot be out of sequence. For example, this is not a legal IF construct:

```
10      ....  
        ENDIF  
        .....  
        IF A=B:GOTO 10
```

The IF statement will look forward through the program to find its corresponding ENDIF statement, and this example implies that the corresponding ENDIF is supposed to be before the actual IF statement. This example will produce an IF: unmatched error.

Write Error

An error in the WRITE command has occurred. This may be because you are writing to the disk and the disk is full.

INDEX



INDEX

- abort
 - 2-6 3-4 4-39 6-9 6-16 7-5,6 8-1 C-1
- AmigaDOS
 - 2-2 2-5 3-6 6-11 7-1 7-6 B-1,2 C-2
- ANIM
 - 3-13 4-38 6-1 6-14,18 8-2 8-17 8-23 8-36 C-4
- arithmetic
 - 4-12 C-3
- array
 - 4-4,5 8-2,3 C-2,3
 - and text strings 4-10,11 6-1,3 8-9 8-38 C-5
 - with file I/O 6-4,5 8-32
 - with keyboard input 6-7,8 8-21,22
 - with color maps 8-18 8-36
- background
 - in BLIT 4-25 6-12,13 8-3
 - in CLEAR 8-8
 - in PEN 8-28
 - in PLANES 6-18
 - in STENCIL 4-21,22
 - in TEXT 4-26,27 5-6 8-12 8-39
- blit
 - 4-24,25 5-2,8 5-10 6-17,18
 - 8-3
 - across resolutions 3-11,12
 - in page-flipping 6-12
 - palette limitations 4-18
- blitdest
 - 4-31 4-33,34 6-14 8-6 8-11
- blitmode
 - 6-14 8-4,6
- blitutil
 - 4-25 5-2,3 6-13
- buffer
 - 3-1,2 3-4,8 3-11,13
 - errors C-2 C-4
 - Font buffer 4-30 8-23
 - Freeing 4-38 8-17
 - in ANIM 8-23 8-36
 - in BLIT 8-3
 - in DISSOLVE 4-20
 - in LOAD 8-22
 - in NEW 8-26

in sound 7-1 7-3,5
in STENCIL 4-21 8-38
in WIPE 4-17,19 8-39
resolution of 8-33
buffers command
8-6
center
text 4-27 8-7 8-24 8-39
overscan screens 6-14 8-30
circle
5-9,10 6-12 8-6 8-8 8-15
clear
8-6 8-8 8-29
close
6-6 6-8 8-8 8-27
colon
3-1 4-3 5-4
color
3-12 4-27 4-36,37 8-8 8-12 8-16 8-18,19 8-27,28 8-34 8-36
comment
4-2 5-9 8-32
compare
6-1,2 6-7 8-9
copy
3-7,8 3-11,12 6-16,17
8-10 B-1,2 C-2
cycle
5-7,8 5-10 8-10 8-16
display
3-2,5 3-7,8 3-11 4-33,34 6-16,17 8-6 8-11 8-16
dissolve
3-11,13 4-18 4-20 8-11 8-16 8-29 C-5
double-buffering
2-3 2-5 3-8 4-13,15 4-24 4-31,35 5-3 5-11 6-12,14 8-2 8-6 8-10 8-16
in ANIM 6-15,18
debug
4-37
defaults
4-2 4-20 4-25,27 5-5,6 6-14 7-3,4 8-1 8-4 8-7 8-12 8-16 8-24 8-28
8-30 8-37 8-39
drawing
4-13 4-31 4-33 4-36 6-11,12 6-14 8-12 8-29
drawmode
4-27 5-6 5-8,9 8-12 8-39

ellipse
4-36 8-13
else
4-7,8 4-16 8-13 8-20
end
4-39 8-13
endif
4-7,8 4-16 5-7,10 6-2 6-4 6-7,8 6-10 6-16,17 8-13,14 8-20 C-6
execute
2-7 4-28,29 6-11 7-1 7-6 8-14 B-2,3 C-2
expression
3-11 4-5 4-7,9 4-11,12 8-1 8-3 8-13 8-16 8-31 8-39 C-2,5
fade
4-16,17 4-19 5-9,10 7-2 7-4,5 8-10 8-15 8-34 8-37
fast ram
3-7,8 3-10,11 4-19,20 4-34,36 4-39 8-22,23 8-25,26 C-5
file
6-3,7 8-8 8-24 8-27 8-32 8-34 8-43 C-3,4 C-6
fill
4-36 6-11,12 8-15 8-28
fixpalette
3-13 8-10,11 8-16 8-42
fonts
3-12 4-26 4-28,30 5-11 8-14 8-17 8-23 8-35 C-2
for
4-8,10 5-4,11 6-3 6-7 6-12 8-16 8-26 C-4 C-6
foreground
4-15 4-26,27 8-8 8-12,13 8-15 8-28,29 8-32 8-39
free
4-38 8-17 8-23
in sound 7-2 7-5
freefont
4-30 8-17
getkey
6-9 8-18
getmap
4-37 8-18
getmouse
6-9,10 8-19 C-5
getpen
4-36 8-19
gosub
4-5,7 4-13 4-15,17 4-19 4-33,34 6-1 6-17 8-19,20 8-33 C-4 C-6

goto
 3-3,4 3-8 4-2 4-5,6 4-13 4-16,17 4-19 4-33 6-7 6-10 6-16,17 8-17 8-20
 C-4 C-6
 if
 4-7,8 4-16 5-7,10 6-2 6-4 6-7,8 6-10 6-16,17 8-13,14 8-20 C-6
 ifkey
 6-9,10 8-21
 ifmouse
 4-29 6-9,10 8-1 8-21 8-30
 IFF
 1-1 2-6 3-1 3-11 4-19 4-26 5-1 6-15 6-18 7-2 8-3 8-22,23 8-28
 input
 4-6,7 4-10 6-2,4 6-6,10 6-17 8-18 8-21,22 8-25 C-1 C-10
 incli
 6-8 8-21 C-1
 keyboard
 6-3 6-7,8 8-21,22 C-1
 kill
 7-2 7-4,6 8-37
 load
 3-1,8 4-2 4-13 4-16,17 4-19,21 4-33 5-2 5-4,6 5-8,9 6-7,8 6-11 6-15,18
 8-7 8-22,23 8-34
 sound 7-2,4 8-37
 loadanim
 6-15,17 8-23
 loadfast
 3-7,8 6-15 8-22,23 C-1 C-5
 loadfont
 4-29,30 5-6 8-17 8-23
 location
 6-4,5 8-24 C-3
 loop
 3-3,4 4-5 4-8,10 5-4 8-16 8-26
 sound 7-2 7-4
 margins
 4-27 8-24 8-39
 memory
 3-5,11 4-38,39 5-3 8-25 C-5
 sound 7-1,2 7-5
 move
 4-36 8-25
 next
 4-8,10 5-4,11 6-3 6-7 6-12 8-16 8-26 C-4 C-6

new
3-7,8 8-26,27 C-5
newfast
3-7 8-26 C-5
numbers
4-3,5 6-1,2 C-5
open
6-3,8 8-8 8-24 8-27 8-32 8-34 8-43 C-2,3
overscan
6-14,15 8-30
page-flipping
3-4,6 3-11 4-1 4-15 4-25 4-34 5-4 6-7 6-12,13 6-18
palette
2-2 3-6,7 3-12,13 4-15 4-18 4-36,37 5-6,10 6-11 6-18
8-4 8-8 8-10,11 8-15,16 8-22 8-26,28 8-33,34 8-42
pause
3-2,6 4-25 5-2 8-28 8-37
pen
4-26,27 4-36 5-6 6-11,12 8-19 8-28
planes
3-5,6 3-12 6-18,19 8-29 8-33
play (sound)
7-2,6
point
4-36 8-29
pointer
6-9,10 8-19 8-30
position
6-13,17 8-29
print
4-5 4-39 6-2 6-4,5 6-7,8 8-20 8-31 8-39 8-43
program flow
4-5,6 4-13 4-16,17 4-19 8-17
quiet
7-2 7-4 8-37
read
6-3,7 8-32 8-43
rect
8-32
return
4-5,7 8-33 C-6
seek
6-3,6 8-27 8-34 C-3

setblack
4-13 4-15,17 4-19 4-33 5-9 8-10 8-15 8-34 8-36
setfont
4-30 5-6 8-23 8-35
setlace
6-1 8-35
setmap
4-37 8-18 8-36
skipanim
6-16,17 8-23 8-36
sound
2-1 2-3,4 2-6,7 6-11 7-1,6 8-14 8-25 8-37
speed
3-2 3-4 3-6 4-24,26 8-28 8-37
stencil
4-21,24 8-4 8-38 C-5
strings
4-10,11 4-38 6-1,4 6-7,8 7-3,6 8-9 8-14 8-20,23 8-25 8-27 8-31
8-37,39 C-1 C-3 C-5
text
4-10 4-26,27 4-30,31 5-1 5-6,11 6-1 8-39
trace
4-37,38 8-41 C-1 C-5
tron
2-7 4-38 8-41
troff
4-38 8-41
variables
3-11 4-3,4 4-7 4-9 4-12 4-38,39 6-1 8-2 8-19,20 8-25 8-33 C-5
wipe
3-8 3-11,13 4-17,22 4-24,25 6-1 6-19 8-4 8-10 8-16 8-29 8-38 8-42
write
4-37 5-1 6-4,5 8-27 8-43 C-3 C-6

Detach Here

REGISTRATION CARD

003

Name _____ Age _____ Sex _____

Address _____ Phone _____

Software Purchased _____ Date of Purchase _____

Where was the software purchased? _____

How much memory does your Amiga have? _____

What peripherals (printers, digitizers, genlocks, etc.) do you have? _____

What magazines do you read? _____

What new products for the Amiga would you like to see? _____

What comments or suggestions do you have for us? _____



PLACE
STAMP
HERE



The Right Answers Group
Box 3699
Torrance, CA 90510



THE DIRECTOR

PROFESSIONAL DISPLAY AND ANIMATION LANGUAGE FOR THE AMIGA™

Envision a creative freedom you've only *dreamed* about...building animations, presentations and computer films that use the *full* power of the Amiga™. Imagine page flipping, color cycling, text generation, even IFF ANIM animations, all combined *at the same time* on the same screen. Now, from the simplest slideshow to the most sophisticated desktop video production, that dream comes true with The Director.

Combine pictures in any resolution from any paint program or digitizer you use with your Amiga. With The Director you write a script to display them in unlimited ways. You can page flip full or partial screens, use drawing commands, create special effects, generate multiple font text, execute AmigaDOS commands, and much more. This is only the beginning of the freedom you give yourself with The Director.

- Use any IFF images, any resolution, any number of colors
- Fades, Dissolves, Blits, Wipes, Stencils
- Page flip full or partial screens
- Preload images, fonts and sounds up to your memory limit
- Flexible script-based structure
- Basic-like vocabulary: For/Next, Gosub/Return, If/Else/Endif
- Arithmetic expressions, random number generator, variables
- Execute AmigaDOS commands from the script
- Text string and file input and output
- Keyboard and mouse interaction
- Digitized soundtrack module
- Supports HAM and overscan
- Supports IFF ANIM playback
- Built-in drawing commands
- No copy protection

And much more...



The Right Answers Group

Box 3699

Torrance, CA 90510

Amiga is a trademark of Commodore-Amiga, Inc. The Director is a trademark of The Right Answers Group.